TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

IMPLEMENTACE PLUG-IN MODULU PRO VÝUKOVÝ SYSTÉM MOODLE

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Jan Hybš**
Vedoucí práce: doc. Ing. Jiřina Královcová, Ph.D.

Liberec 2014

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

IMPLEMENTATION OF PLUG-IN MODULE FOR THE MOODLE SYSTEM

Diploma thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Jan Hybš**
Supervisor: doc. Ing. Jiřina Královcová, Ph.D.

Liberec 2014

Tento list nahraďte
originálem zadání.

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Chtěl bych tímto poděkovat všem, kteří se podíleli na této diplomové práci. Byla to především má rodina, která mi poskytla podporu finanční a psychickou bez které bych nebyl schopen práci dokončit. Děkuji přátelům, učitelům, studentům a všem dalším, kteří mi pomohli s realizací této práce. Dále bych chtěl poděkovat Ing. Igoru Kopetschkemu, který mi pomohl v realizaci této práce užitečnými konzultacemi.

Závěrem bych chtěl poděkovat zejména vedoucí práce doc. Ing. Jiřině Královcové, Ph.D. za poskytnuté konzultace, užitečné rady a za možnost zabývat se touto problematikou.

Abstrakt

V této práci je popsána problematika testování dovedností studentů v předmětech, které se zabývají programováním a algoritmizací. Celý proces začíná vytvořením algoritmické úlohy. Vytváření úloh je v kompetenci pedagoga, který specifikuje problematiku úlohy a definuje vstup a výstup úlohy. Studenti na základě specifikace vytváří algoritmy, které řeší tuto úlohu. Řešení studentů jsou zpracována a otestována. Výsledky testů poté určují finální známku řešení. Cílem této práce je seznámit se s tvorbou modulů pro systém Moodle a vytvořit modul, který bude umožňovat specifikaci programových úloh. Pomocí tohoto modulu bude dále možné zasílat řešení na úlohy a kontrolovat efektivnost a korektnost odevzdaných řešení. Modul dále musí podporovat určení podobnosti odevzdaných řešení pro odhalení plagiátů.

V textu je představen systém, který napomáhá procesu testování dovedností studentů. Vytvořený systém nese název CoDiAna (**Code Diagnosis Analyzer**) a skládá se ze dvou hlavních částí. První částí je modul CoDiAna pro výukovou platformu Moodle zajišťující vytváření a správu algoritmických úloh. Specifikace úlohy musí obsahovat hodnotící kritéria pro časovou a paměťovou náročnost a další parametry související s hodnocením. Modul poskytuje prostředky, které zjednodušují proces vytváření programových úloh. Specifikaci vstupních souborů lze provádět pomocí generátoru vstupních souborů. Výstupní soubor může být také vygenerován, pokud je k úloze přiloženo řešení pedagoga. Zpracování tohoto řešení detekuje přibližnou časovou a paměťovou náročnost úlohy. Pomocí modulu lze prohlížet dosažené výsledky studentů, a to ve formě statistických grafů nebo detailního výpisu. Vytvořený modul CoDiAna dále umožňuje vyvolat kontrolu úlohy nebo řešení a detekovat tak plagiátorství.

Druhou klíčovou částí systému CoDiAna je Exekutivní aplikace, která slouží pro zpracování odevzdaných řešení a určení podobnosti mezi řešeními. Exekutivní aplikace podporuje dynamické načítání dalších modulů, které přidávají podporu dalších programovacích jazyků.

Teoretická část práce vysvětluje základy výukové platformy Moodle. V textu jsou vylíčeni uživatelé a role v systému Moodle a s nimi související pravomoci a oprávnění. Je zde věnována kapitola tvorbě modulů pro systém Moodle a je nastíněna konvence psaní pro vývojáře systému. V praktické části je dále popsána struktura a princip funkce celého systému spolu s výsledky testů, které byly prováděny pro určení optimálních algoritmů. Spolu se systémem byly také vytvořeny příručky (pro vývojáře a pro uživatele systému CoDiAna), které usnadní práci se systémem.

Klíčová slova: algoritmické úlohy, syntaktická analýza zdrojových kódů, systém Moodle, systém CoDiAna, programování

Abstract

This work describes problem of testing student's proficiency in subjects that deal with programming and algorithmization. The whole process begins with the creation of algorithmic tasks. Creation of these tasks is teacher's responsibility. Teacher describes tasks problem and defines input and output of the task. Students then create algorithms based on problem's specification solving given task. Student's solutions are prepared and tested. The result of these tests determine solution's final grade. The main objective of this work is to research module creation in system Moodle and create such module which will be able to specify algorithmic tasks. This module will allow sending task's solutions and determine their correctness and efficiency. The module must support detection of plagiarism by estimating solutions similarity.

The text introduces system which simplifies process of testing student's proficiency. The system is called CoDiAna (**Co**de **Di**agnosis **Ana**lyzer) and consists of two main parts. The first part is module CoDiAna for system Moodle ensuring the creation and management of algorithmic tasks. Task specifications must include evaluation criteria for time and memory requirements and other options related to the evaluation process. The module provides tools simplifying process of task creation. Input file specification can be done using input file generator. Output file can also be generated when teacher's solution is uploaded. Processing of this solution estimates time and memory requirements. Using this module teacher can view student's results in form of statistical graphs or detailed listing. Module CoDiAna can invoke examination of task or solution to detect plagiarism.

The second key part of system CoDiAna is executive application which processes sent solutions and determines similarity among solutions. Executive application allows dynamic module loading which adds support of additional programming languages.

The theoretical part explains the basics of e-learning platform Moodle. Paper describes users and roles in the Moodle system and related topic of capabilities and permissions. Text includes chapter dedicated to module creation. In this chapter is also outlined programming convention for developers. Practical part of the text describes structure of system and how system works. There are also results of tests that were carried out to determine the optimal algorithms. Two manuals were made along with the system (for developers and for users) which provides helpful hints when using system CoDiAna.

Keywords: algorithmic tasks, parsing the source code, Moodle system, CoDiAna system, programming

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době je použití informačních technologií a výpočetní techniky stále frekventovanější. Objevují se nová uplatnění těchto odvětví a mohou zefektivňovat stávající řešení v mnoha oblastech. Jednou z nich je školství, kde jsou za pomoci informačních technologií vytvářeny výuková prostředí, která nabízí nemálo prostředků usnadňující práci jak pedagogům, tak studentům. Výukové platformy slouží pro vytváření a sdílení obsahu skrze celosvětovou internetovou síť. Přináší do školství e-learning, výuku, která využívá výpočetní techniku spolu s internetem. Systém Moodle je jednou z těchto výukových platforem a je používán na Fakultě mechatroniky, informatiky a mezioborových studií na Technické univerzitě v Liberci. Využití informačních technologií nemusí vždy přinést zlepšení daného problému. Mnohdy je nutné určit kompromis mezi požadovanou kvalitou výsledků a poskytnutým časem, pro vyřešení problému. Teoreticky je však možné s neomezenými časovými a paměťovými prostředky vyřešit každý algoritmický problém.

Tato práce zasahuje do odvětví zpracování a analýzy zdrojových kódů a snaží se tuto problematiku usnadnit a přesto poskytnout kvalitní výsledky. Cílem práce je seznámit se s realizací modulů pro systém Moodle a vytvořit modul řešící problematiku správy programovacích úloh, které slouží pro otestování dovedností studentů. Základním prvkem této problematiky je programátorská úloha, kterou definuje pedagog v předmětech s programovací tématikou. Úlohy jsou definované zadáním a specifikací vstupu a výstupu. Studenti na základně zadání úlohy odevzdávají svá řešení v podobě algoritmu, prostřednictvím modulu pro systém Moodle. Vytvořený systém by měl umožňovat specifikaci zadání úloh, zpracování přijatých řešení a měření hodnotících veličin. Měl by také podporovat ověření správnosti řešení porovnáním vygenerovaného výstupního souboru s referenčním výstupním souborem úlohy. Dále pak možnost testování efektivity řešení a analýzy podobnosti řešení. Systém musí řešení studentů ohodnotit a navrhnout příslušnou známku. Další činností systému je detekce duplicitních řešení, tedy nalezení plagiátů mezi odeslanými řešeními.

Doposud byla kontrola úloh v předmětech s programovací tématikou prováděna různými způsoby. První způsob je ruční kontrola, kde musí pedagog jednotlivá přijatá řešení spustit, otestovat a nadále ohodnotit známkou. Tento způsob produkuje nejpřesnější výsledky, ale za cenu velké časové náročnosti. Dále je možné použít automatizované systémy, které mohou problematiku zjednodušit. Použití takovýchto systémů může přinést značnou časovou úsporu, ale výsledky mohou být méně přesné. V rámci bakalářské práce [6] byl obdobný systém realizován. Samotný systém však není propojený s e-learningovým portálem TUL ani možnost propojení nenabízí.

Textová zpráva je rozdělena do šesti hlavních kapitol. Nejprve jsou prozkoumány možnosti výukové platformy Moodle, především možnosti vytváření nových modulů pro tento systém. Dále je v textu popsán systém CoDiAna, jeho struktura a princip funkce. V práci jsou věnovány kapitoly hlavním částem systému, tj. modulu CoDiAna a Exekutivní aplikaci Java. Další kapitola popisuje bezpečnostní aspekty systému CoDiAna. Závěrečná kapitola se zabývá testováním funkčnosti vytvořeného systému.

1 Teoretická část

1.1 Moodle

Moodle (Modular Object-Oriented Dynamic Learning Environment) je softwarová platforma, která pedagogům a studentům poskytuje prostředky pro tvorbu výukového prostředí. Platforma Moodle je poskytována jako open-source, je tedy zcela zdarma a navíc velice přizpůsobitelná – nabízí mnoho rozšíření. Základní myšlenou systému Moodle je sociálně konstruktivní přístup ke vzdělání, které považuje vzdělání nebo myšlenku jako neustále se měnící interakcí uživatelů [2].

1.1.1 Struktura

Základním stavebním prvkem systému Moodle jsou `kurzy`, které obsahují studijní materiály a činnosti. `Studijním materiálem` může být prezentace, odkaz webové stránky nebo například videozáznam přednášky. `Činnosti` v systému určují aktivitu, která produkuje výsledky. Mohou to být například testy, úkoly nebo ankety.

Systém obsahuje hlavní funkční jádro, složené z malých celků, spravující uživatelské role, pravomoci, moduly, kurzy a další důležité základní oblasti. Na tento hlavní celek jsou napojeny desítky modulů, které dále rozšiřují možnosti systému. To vše dohromady tvoří systém Moodle.

1.1.2 Pravomoci a oprávnění

Terminologie systému Moodle definuje pravomoci jako možnost provádět se systémem určitou činnost. Každá pravomoc se vztahuje k určité operaci. Oprávnění úzce souvisí s pravomocemi a je definováno jako pravomoc, která je zakázána nebo povolena určité roli v systému. Každý kurz má základní sadu pravomocí, kterou lze upravovat. Pravomoci jsou definované v každém studijním materiálu nebo činnosti. Vývojáři pak zodpovídají za výchozí hodnoty oprávnění nově vytvořených instancí kurzů nebo objektů v kurzech.

Příkladem pravomoci může být akce `hodnotit test`. Oprávnění pro pedagoga této pravomoci bude povoleno, ale pro studenty bude tato možnost

zakázána. Pokud má však daný pedagog oprávnění k přidělování jiných oprávnění, může vybraného studenta pověřit a udělit mu oprávnění k „hodnocení testu“.

1.1.3 Uživatelé a role

V systému Moodle jsou definované různé role, jako například pedagog nebo student, ale myšlenka systému Moodle umožňuje pověřit uživatele různými pravomocemi. Nejsou tedy přesně definované možnosti jednotlivých rolí. Správu pravomocí určuje buď pedagog, který řídí daný kurz nebo administrátor, který má neomezené pravomoci. Systém rolí spolu se systémem pravomocí a oprávnění tvoří komplexní, ale dynamickou strukturu, která umožňuje efektivně měnit přístupy jednotlivých uživatelů.

1.1.4 Rozšíření a moduly

Dynamická struktura systému umožňuje přidávání další nestandardní funkcionality. V systému Moodle je řada různých druhů modulů. Mohou to být například moduly činností nebo různé filtry a editory. Každý modul musí splnit základní požadavky, aby mohl být přidán do systému. Požadavky pro moduly činností jsou následující:

- Musí obsahovat určité základní pravomoci a jejich oprávnění. Vývojář poté může přidat další pravomoci, které budou v rámci modulu používány.
- Musí vždy existovat minimálně jedna tabulka v databázi systému.
- Musí být implementovány základní funkce, které slouží pro správu jednotlivých instancí modulu.

1.1.5 Konvence psaní pro vývojáře systému Moodle

Vývoj nového software pro systém Moodle by měl dodržovat jistá pravidla. Jedná se především o pravidla pro názvy proměnných, tříd, funkcí a souborů. Pokud je vyvíjen nový modul, musí dodržovat prefix u většiny metod nebo funkcí. Systém Moodle podporuje multijazyčnost a každý nový modul by měl tento fakt respektovat a být tak připraven pro podporu více jazyků. Vývojář musí specifikovat slovníkové klíče v přesně definovaných souborech, které pak obsahují výslednou frázi. Neměl by tedy vypisovat přímo výslednou frázi [2]. Následující

ukázka slouží k jednoduchému výpisu v jazyce PHP, tento přístup by neměl být používán, pouze jsou-li vypisována data z databáze.

```
echo 'Vítejte v modulu CoDiAna';
```

Vývojář by měl využít funkce systému a slovníkový klíč. Tento klíč je univerzální a na základě zvoleného jazyku systému Moodle se načte výsledná fráze ze souboru, který obsahuje všechny slovníkové fráze a hodnoty pro daný jazyk. Následující ukázka ukazuje korektní způsob, který by měl být používán při vývoji nového modulu.

```
print_string ('welcome_message', 'codiana');
```

1.1.6 Tvorba nového modulu

Struktura modulu se může různit v závislosti na verzi systému Moodle. Rozlišují se dvě základní struktury pro verzi 1.9 a pro verze 2.x. Oficiální dokumentace systému Moodle popisuje, jak vytvořit nový modul pro verzi 1.9, ale značná část návodu platí i pro verze 2.x. Prvním krokem při vytváření modulu je stažení archivu, který obsahuje ukázkové soubory pro tvorbu nového modulu. Všechny soubory balíčku jsou popsány a je vysvětlena jejich funkčnost. Jedná se o několik základních souborů, které tvoří nový modul s názvem `newmodule`. Mezi důležité soubory patří například struktura databáze, knihovní funkce nebo definice pravomocí a oprávnění [2].

Struktura databáze nového modulu je uložena v souboru `db/install.xml`. Jsou zde definovány všechny tabulky modulu. Tento soubor je načten v průběhu instalace a na základě obsahu je vytvořena požadovaná struktura databáze Moodle. Soubor knihovních funkcí (`lib.php`) obsahuje základní funkce pro tvorbu modulů. Názvy funkcí musí začínat názvem modulu. Soubor pravomocí a oprávnění (`access.php`) definuje všechny pravomoci, které mohou být v modulu použity. Ke každé pravomoci jsou definovány i oprávnění uživatelů systému, typ pravomoci a potencionální rizika dané pravomoci. Oprávnění mohou dědit vlastnosti jiných oprávnění.

Instalace nového modulu se skládá z několika kroků. Nejprve je nutné zkopírovat všechny soubory nového modulu do příslušného adresáře

(`/mod/nazev_modulu`). Po přihlášení do systému v roli administrátora je spuštěn průvodce instalací nového modulu. Administrátor musí nejprve potvrdit aktualizaci databáze. V dalším kroku, pokud tuto činnost nový modul podporuje, administrátor vyplní základní nastavení modulu. Instalace je dokončena a nový modul je možné začít používat.

Moduly jsou označeny unikátním názvem a obsahují mimo jiné verzi modulu. Tato verze je užitečná například při definování závislostí mezi moduly. Další využití verze je například při aktualizaci modulu. Vývojář může provádět aktualizační rutinu, která se může různit pro určité verze modulu. Každý nově přidaný modul je možně odebrat v administrátorské sekci.

2 Systém CoDiAna

2.1 Činnost systému

Systém CoDiAna[1] podporuje praktickou část v předmětech zabývajících se programováním. Znamená to, že pedagog může pomocí tohoto systému testovat znalosti a zkušenosti studentů v problematice programování. Hlavní součástí systému jsou programovací úlohy. Tyto úlohy popisují určitou problematiku, kterou se studenti snaží vyřešit. Každá programovací úloha má definované vstupní a výstupní požadavky a studenti vytváří a zasílají algoritmy produkující určité výsledky. Všechny správné výsledky jsou systémem ohodnoceny na základě časové a paměťové náročnosti. Optimálnější algoritmy mohou dostat vyšší bodové ohodnocení. V kompetenci pedagoga je určení hodnotících kritérií. V některých případech není žádoucí nejrychlejší výsledek nebo výsledek s nejnižší paměťovou náročností, ale výsledek, který využívá různé návrhové vzory nebo metody. Pedagog tak může efektivně ovlivňovat výsledky studentů a nemusí algoritmy kontrolovat nebo dokonce ani spouštět manuálně. Mezi správnými řešeními však mohou být plagiáty. Automatizovanou kontrolu všech přijatých řešení provádí systém CoDiAna a snaží se nalézt podobné algoritmy. Pedagogova spoluúčast je vyžadována pouze pro algoritmy, které systém označil jako podezřelé z plagiátorství.

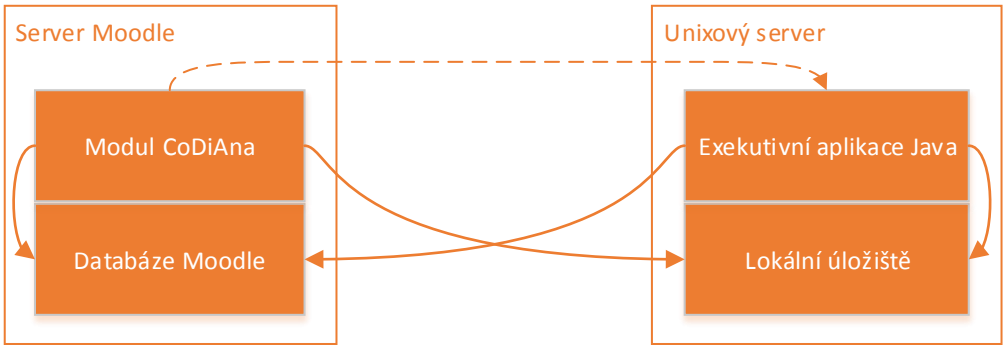
Výsledný systém poskytuje prostředky pro vytváření programovacích úloh a umožňuje odevzdání a zpracování řešení. Dále nabízí prostředky pro detekci duplicitních řešení. Všechny možnosti systému však neprovádí modul pro systém Moodle, časově náročnější nebo potencionálně nebezpečné operace jsou přenášeny na jiný server, kde jsou za pomoci Exekutivní aplikace zpracovány. Systém CoDiAna neklade žádná omezení na podporu programovacích jazyků, umožňuje zpracování řešení v libovolném, serverem podporovaném, jazyce.

[1] akronym pro Code Diagnostic Analyzer

2.2 Struktura systému

Celý systém se může skládat až ze dvou serverů, kde oba servery vykovávají jinou činnost. Na prvním serveru je nainstalován systém Moodle a modul CoDiAna. Tento server přijímá požadavky, ukládá nastavení úloh a předem definovaným impulsem může upozornit jiný server na určité události. Slouží tedy pouze pro správu úloh. Druhý server, na kterém je nainstalován unixový operační systém a Exekutivní aplikace, slouží pro zpracování úloh. Jedná se především o kompilaci, spuštění a ukládání výsledků.

Oba servery spolu mohou komunikovat pomocí zabezpečeného protokolu SSH. Systém CoDiAna nevyžaduje dva servery, umožňuje použití jen jednoho serveru, který však musí splňovat všechny požadavky. Nastavení se pak provádí v konfiguračních souborech a v globálním nastavením modulu CoDiAna. Použití dvou serverů má určité výhody, například přesnější naměřené výsledky řešení, neboť výpočetní server obstarává pouze zpracování a není ovlivněn asynchronními událostmi. Izolace serverů zvyšuje bezpečnost, jelikož na výpočetním serveru jsou provozovány potencionálně nebezpečné operace. Možnost použití dvou serveru umožňuje využití jiného přímo specializovaného výpočetního serveru pro zpracování [1].



Obrázek 1: Struktura systému CoDiAna

Obrázek 1 znázorňuje strukturu systému Moodle a komunikace mezi zmíněnými servery. Server Moodle obsahuje činnostní modul CoDiAna. Na serveru se také nachází databáze systému Moodle, která obsahuje nejenom informace systému Moodle, ale i všech nainstalovaných modulů. Modul CoDiAna komunikuje s unixovým serverem. Tento server obstarává zpracování výsledků a obsahuje Exekutivní aplikaci a lokální úložiště. Komunikace mezi servery probíhá prostřednictvím úložišť, databázového úložiště na serveru Moodle a lokálního úložiště na unixovém serveru. Modul CoDiAna přijímá požadavky od uživatelů systému Moodle a zpracovává pouze některé z nich. Do databáze jsou uloženy požadavky, které není modul CoDiAna schopen zpracovat. Exekutivní aplikace slouží pro zpracování právě těchto požadavků. Na základě údajů z databáze postupně vyřizuje požadavky a zapisuje výsledky do databáze.

3 Modul CoDiAna

Výsledný modul patří mezi činnostní moduly, produkuje tedy určité výsledky. Modul umožňuje vytváření programovacích úloh. U každé úlohy lze specifikovat sadu vlastností. Tyto vlastnosti mohou ovlivňovat běh Exekutivní aplikace. Princip modulu je následující: Je vytvořena programovací úloha, která má specifikovány vstupní a výstupní soubory. Studenti poté zasílají řešení, které Exekutivní aplikace vyhodnocuje. Vyhodnocení, které je ovlivněno nastavením úlohy, je sestaveno z měřených vlastností spuštěného řešení. Bere se v potaz výstup řešení, který musí být totožný s referenčním výstupem. Jako další faktor, který je součástí hodnocení, je čas běhu řešení a průměrná paměťová náročnost spuštěného řešení. Všechny měřené hodnoty poté určují výsledek řešení. Pokud některá z vlastností nesplňuje minimální požadavky, například maximální čas běhu, je řešení hodnoceno jako nedostatečné.

Při vývoji modulu byl použit převážně programovací jazyk PHP, neboť je základním jazykem pro systém Moodle. Další použitý jazyk v modulu, tentokrát značkovací, je XML. Je použit především pro definování struktury databáze. Systém Moodle vyžaduje definici struktury databáze modulů v tomto jazyce. Obsahuje nástroje, které ulehčují vytváření XML popisu databáze. Jedná se například o editor XMLDB, který umožňuje jednoduše vytvářet tabulky a sloupce.

3.1 Struktura modulu

Činnostní modul CoDiAna využívá dvou typů úložišť. Jedná se o databázi a standardní souborové úložiště na serveru. To, zda budou data uložena do databáze nebo do souboru, již určuje jejich charakter. Výsledná databázová struktura musí splňovat základní pravidla systému Moodle pro činnostní moduly. Jedná se o jednoduchá pravidla, která nijak neomezují vývojáře, pouze definují konvenci psaní. Každý činnostní modul musí v databázovém systému Moodle obsahovat alespoň jednu tabulku. Tabulka nese stejný název jako označení modulu a musí obsahovat alespoň jeden sloupec `id`, který jednoznačně identifikuje instanci daného modulu. V případě modulu CoDiAna existuje základní tabulka `codiana`, která obsahuje všechny úlohy v systému. Povinná tabulka může dále obsahovat

další sloupce, které už nejsou nijak limitované. Moduly mohou obsahovat další tabulky, které napomáhají v realizaci problematiky daného modulu. Konvence systému Moodle doporučuje primární klíč tabulek jako sloupec s názvem id. Sloupec by měl být datového typu `integer` a zároveň by měla být nastavena vlastnost `autoincrement`[2].

3.1.1 Databázové úložiště

Výsledná databázová struktura modulu CoDiAna se skládá z pěti tabulek. První již zmiňovaná tabulka `codiana`, slouží pro ukládání úloh, zejména jejich nastavení. Každý řádek reprezentuje jednu úlohu. Obsahuje název a specifikaci úlohy (popis, ukázkové vstupy a výstupy, název hlavního souboru, dovolené programovací jazyky a další vlastnosti). V této tabulce jsou dále definovány všechny limity úlohy (časové, paměťové, počty řešitelů a pokusů) a rozmezí dostupnosti úlohy (od kdy je dostupná a do kdy je úloha dostupná). Jsou zde uloženy i způsoby, jakými jsou porovnávány výstupní soubory nebo také metody, na základě kterých jsou vybrány hodnocené pokusy studentů. Každá úloha obsahuje informace o tom, kdy byla spuštěna detekce duplicitních řešení a jakým výsledkem skončila. Tabulka `codiana` obsahuje celkem 27 sloupců.

Druhá tabulka v modulu nese název `codiana_attempt` a slouží pro ukládání jednotlivých pokusů od řešitelů. Pokud tedy student zašle řešení je vytvořen záznam, který drží informace o tomto pokusu. Zpočátku je většina polí záznamu převážně prázdná, ale po zpracování jsou naplněna informacemi o běhu či měření daného pokusu. Tabulka obsahuje údaje o tom, kdy byl pokus zaslán, v jakém jazyce je zdrojový kód nebo o kolikátý pokus studenta se jedná. Jsou zde uloženy detaily časové a paměťové náročnosti, korektnost výstupního souboru i finální výsledek. Pokud při zpracování nastane chyba, je v poli `poznámka` její popis. Pedagog má možnost upravit tuto hodnotu, což umožňuje předávání zpráv řešiteli. Důležitou vlastností záznamu je také status pokusu, tj. v jakém stavu se pokus nachází. Výchozí hodnota odpovídá stavu `čekání na zpracování`, neboť je záznam vytvořen, když řešitel nahraje zdrojový kód na server. Postupem se hodnota změní v závislosti na korektnosti zaslaného řešení. Pedagog má možnost

[2] Autoincrement automaticky generuje unikátní hodnotu pro primární klíč tabulky. Hodnota se postupně inkrementuje při vložení nového záznamu.

měřit hodnoty úlohy zasláním referenčního řešení. Tato akce vyústí ve vytvoření nového záznamu v této tabulce. Záznam však obsahuje příznak indikující nestandardní pokus. Tento pokus je viditelný pouze pro pedagoga a není započítán do celkového počtu pokusů uživatele (například pokud se pedagog přepne do role studenta, a již využil možnost měření hodnot, nebude v celkovém počtu pokusů započítán nestandardní pokus).

Pro komunikaci mezi modulem CoDiAna a Exekutivní aplikací Java slouží tabulka s názvem `codiana_queue`. Každý záznam tabulky reprezentuje požadavek pro Exekutivní aplikaci. Požadavek může být typu zpracování řešení, měření hodnot nebo detekce duplicitních řešení. Tabulka obsahuje atribut `typ`, ve kterém je uložen kód typu požadavku. Dále každý záznam obsahuje identifikační číslo úlohy, uživatele, který požadavek vytvořil a v některých případech `id` pokusu a `id` uživatele, se kterým pokus souvisí – vše závisí na typu požadavku. Pokud je vyžadována kontrola duplicitních řešení nad určitým řešením, je nutné znát `id` tohoto uživatele a `id` souvisejícího pokusu. Při kontrole duplicity nad celou úlohou obsahují nepoužívané pole hodnotu `NULL`. Tabulka má charakter metody `FIFO`[3] z pohledu řešitelů. Student, který pošle řešení dříve než jiný student, bude mít výsledky zpracovány dříve. Tabulka však také obsahuje sloupec `priorita`, který může chování pozměnit. Vyšší prioritu ke zpracování obdrží od systému pouze pedagog nebo jiný pověřený uživatel. Exekutivní aplikace po zpracování požadavků záznamy odstraní, aby nebyly zpracovány znovu.

Pro skladování výsledků ohledně plagiátorství slouží čtvrtá tabulka `codiana_plags`. Struktura tabulky obsahuje pouze informace, mezi kterými dvěma studenty nastala vyšší než povolená shoda algoritmů. Hodnota podobnosti uložená v tabulce je v rozmezí od 0 do 100, kde 100 značí úplnou shodu. Další nepovinný atribut tabulky poznámka, může uchovávat bližší informace ohledně naměřené podobnosti. Studenti nemají přístup k těmto výsledkům – přístup má pouze pedagog nebo uživatel pověřený pedagogem.

Poslední tabulka `codiana_language` slouží pro uchovávání nainstalovaných a podporovaných programovacích jazyků. Struktura je velice jednoduchá – záznam

[3] First in, First out (Fronta)

obsahuje název programovacího jazyka (i jeho verzi) a dále příponu souboru pro daný programovací jazyk. Údaje v této tabulce jsou viditelné při vytváření nebo úpravě úlohy, kdy je možné zvolit, jaké programovací jazyky mohou být pro danou úlohu použity. Tento číselník je možné upravovat přímo v databázi. Aby Exekutivní aplikace umožňovala zpracování daného programovacího jazyka, je nutné, aby v konfiguračním souboru byly záznamy o podpoře jazyku pod stejným názvem jako v této tabulce.

3.1.2 Souborové úložiště na serveru

Na server jsou ukládány soubory úlohy v přesně definované struktuře. Při zpracování řešení jsou používány samotné soubory. Jedná se zejména o vstupní a výstupní soubory a soubory obsahující zdrojové kódy. Uložení souborů do databáze má určité výhody (například přenositelnost, zapouzdřenost, bezpečnost), ale výhody pro uložení některých dat přímo do souborů převažují. Všechny soubory na serveru jsou uloženy do složek a podsložek dle určitých pravidel tak, aby byla možná jednoduchá správa souborů na serveru. Struktura uložení souborů začíná v definovaném kořenovém adresáři. Pokud zašle student řešení na určitou úlohu, jeho soubor bude uložen do následujícího adresáře:

```
./kořenový-adresář/task-id-ulohy/user-id-uživatele/curr/
```

Adresářová struktura obsahuje složky a podsložky, které označují danou úlohu a daného uživatele. Všechny pokusy studenta jsou uloženy v zip archivu, jehož název obsahuje pořadové číslo pokusu studenta. Sdílené soubory úlohy (referenční vstupy a výstupy) jsou uloženy v kořenovém adresáři příslušné úlohy. Takto definovaná struktura umožňuje jednoduchou správu všech souborů.

Systém Moodle obsahuje obdobný sytém pro správu souborů na serveru, ale jeho použití by zkomplikovalo přístup Exekutivní aplikaci. Systém Moodle obsahuje API pro tento pokročilý systém správy souborů pro jazyk PHP, avšak není k dispozici pro jazyk Java. Ukládání a načítání souborů vyžaduje argumenty, které lze získat v jazyce Java pouze komplikovanou cestou. Každý uložený soubor je závislý na kontextu, ve kterém byl vytvořen. Kontext může být dále závislý na jiných kontextech, struktura se tedy komplikuje. Z těchto důvodů byl vytvořen vlastní systém, který umožňuje jednoduchou práci se soubory v jazyce PHP i Java.

3.2 Synchronizace konstant

Celý systém obsahuje různé číselníkové hodnoty, používány Exekutivní aplikací Java i modulem CoDiAna. Použití číselníků může usnadnit a zpřehlednit práci. Je ale nutné, aby byly hodnoty číselníku stejné. Je k dispozici celkem šest číselníků, jedná se o:

1. stavy úloh
2. stavy pokusů
3. stavy kontroly plagiátorství
4. módy porovnávání výstupních souborů
5. módy výběru hodnocených řešení
6. typ požadavku modulu CoDiAna

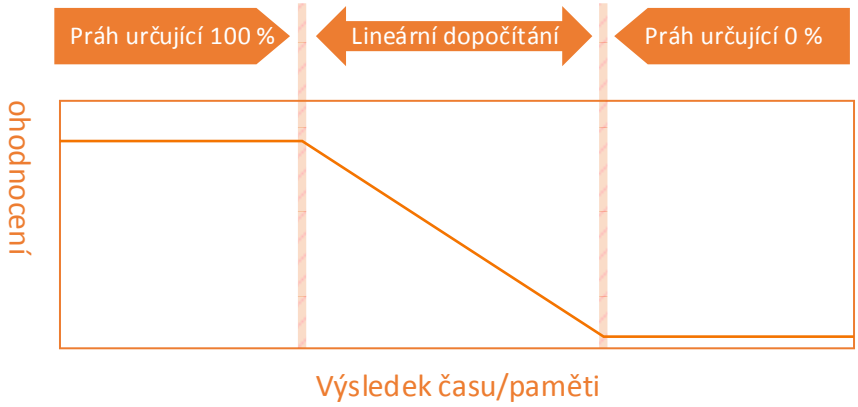
Tyto hodnoty jsou v modulu CoDiAna realizovány pomocí tříd, které obsahují konstanty s žádanými hodnotami. Exekutivní aplikace CoDiAna využívá výčtový typ `Enum`, který v konstruktoru přebírá s potřebnou hodnotu.

3.2.1 Adresace souborů na serveru

3.3 Nastavení úlohy

Základní nastavení určuje jméno úlohy a hlavní soubor úlohy – název souboru/třídy, který bude studenty odevzdáván. Mezi další základní nastavení patří popis, kde je vysvětlena problematika úlohy a dále metody výběru hodnoceného řešení a porovnání výstupů. Metoda výběru hodnoceného řešení určuje, které řešení se vybere při známkování. Metoda porovnání výstupů umožňuje specifikovat způsob, jakým budou porovnávány výstupní soubory (přísnost porovnávání).

Každá úloha může mít specifikovaný začátek a konec. V průběhu, pokud je úloha označena pedagogem za aktivní, mohou studenti posílat svá řešení. Časová a paměťová náročnost je definována dvojicí prahů. První hodnota určuje hranici, do které je ohodnocení 100 %. Druhá hodnota již určuje hranici, po které je ohodnocení 0 %. Ohodnocení mezi prahy je lineárně dopočítáno. Obrázek 2 znázorňuje určení výsledku časové a paměťové veličiny.



Obrázek 2: Graf určení hodnoty veličiny

Pedagog může dále specifikovat hodnotu atributu `maximum uživatelů`, která omezuje počet řešitelů v dané úloze. Atribut `maximum pokusů` může limitovat studenty, kteří řeší danou úlohu. Každý student má poté omezený počet pokusů na odevzdání řešení. Pokud tedy první odeslané řešení studenta skončilo chybou, může řešení opravit a poslat ho znovu.

Důležitými vlastnosti úlohy jsou ukázkové vstupy a výstupy doplňující textový popis úlohy. Specifikace definuje problematiku úlohy a popisuje vstupy a výstupy. Ukázkové vstupy a výstupy řešiteli znázorňují, v jakém formátu mají data být.

3.3.1 Vstupní a výstupní soubory úlohy

Možnosti modulu jsou rozdílné pro studenty a pedagogy (za předpokladu, že jsou oprávnění ponechána v původním stavu). Ke každé vytvořené úloze je nutno definovat vstupní a výstupní soubory. Modul CoDiAna umožňuje nahrání obou souborů na server. Vstupní soubor je možné také vygenerovat pomocí `generátoru vstupních souborů`.

Pedagog definuje jednoduchou strukturu vstupního souboru pomocí pěti základních elementů:

1. sekce (reprezentuje opakující se činnost)
2. číselná kroková proměnná (závislá na sekci)
3. číselná náhodná proměnná (určená minimem a maximem)
4. libovolný řetězec
5. nový řádek

U každé číselné proměnné je možno specifikovat formát, v jakém bude zobrazena. Tento formát je standartní formátovaná konstrukce, kterou lze vidět u mnoha programovacích jazyků, například Java, C, Python a další. Následující příklad demonstruje generování náhodného času:

```
[hodiny][:][minuty][\n]
```

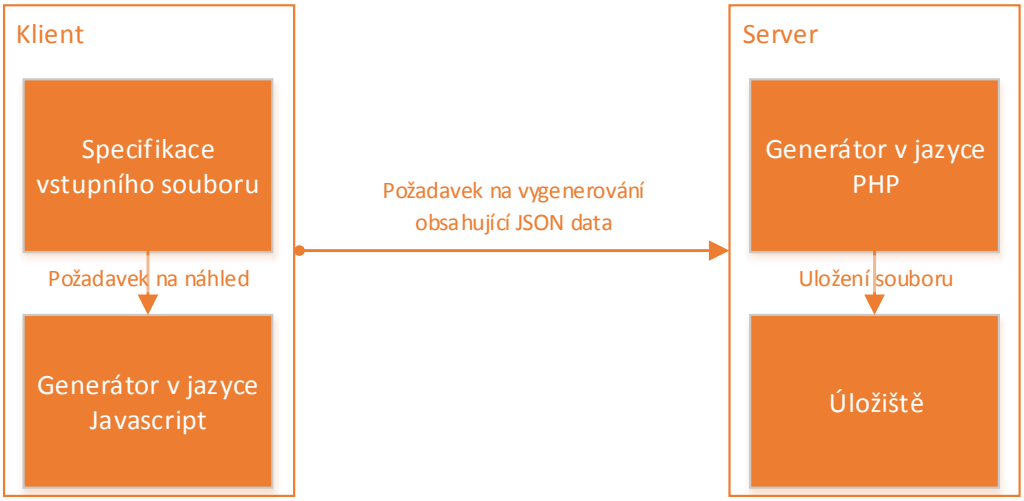
Kde proměnná hodina je náhodná číselná proměnná od `0` do `23`. Proměnná minuty je opět náhodné číslo, ale s rozpětím od `0` do `59`. Formátový řetězec obou čísel je `%02d`, každé vygenerované číslo kratší než dva znaky je doplněno nulou. Ukázka zahrnuje i proměnnou dvojtečku a nový řádek. Výsledek může tedy vypadat takto:

```
08:35

```

Složitější vstupní soubory už není možné generovat nebo by jejich časová realizace byla náročnější v porovnáním s napsáním jednoduché aplikace, která by žádaný vstup generovala. Výhodou použití tohoto generátoru je, že pedagogem vytvořená struktura je uložena a je jí možné později upravovat (například pozměnit formát dat). Pedagog navíc může vidět ukázku vygenerovaného vstupního souboru. Princip funkce generátoru je následující: Na straně klienta pedagog prostřednictvím webového rozhraní postupně generuje žádaný vstup. Vygenerovaná struktura je uložena v paměti a obsahuje objekty s definovanou strukturou. Pro vygenerování souboru je nutné potvrdit formulář, ve kterém se následně vytvořená struktura převede na formát JSON. Dále jsou data odeslána na server. PHP skript zpracuje přijatá data a znovu z nich vytvoří objekty a vygeneruje

vstupní soubor. K dispozici je náhled vstupního souboru, který je automaticky aktualizován při každé změně. Generování náhledu souboru je prováděno na straně klienta, kde je algoritmus generování duplikován (pouze je použit jiný programovací jazyk a je omezena velikost náhledu). Není tedy nutné posílat požadavek na server kvůli každé změně, náhled je aktuální. Celý proces znázorňuje Obrázek 3 uvedený níže. Hodnota náhodných číselných proměnných je generována pseudonáhodným generátorem. Při změně struktury jsou tedy hodnoty těchto proměnných zachovány.



Obrázek 3: Komunikace při generování vstupního souboru

Za předpokladu, že je již vstupní soubor uložen na serveru, je možné výstupní soubor vytvořit pomocí korektního algoritmu na danou úlohu. Pedagog na server nahraje správné řešení, které bude spuštěno a na základě vstupního souboru bude vygenerován výstupní soubor. Pedagog má jistotu, že výstupní soubor je opravdu korektní a reaguje na vstupní soubor uložený na serveru. Tento způsob přináší výhody umožňující například měření různých hodnotících kritérií.

3.3.2 Hodnotící kritéria úlohy

V úvodní kapitole byly zmíněny časové a paměťové limity pro každou úlohu. Tyto hranice je nutné určit u každé úlohy, od které je očekáván hodnocený výsledek. Jak ale tyto prahy zjistit? Jednou z možností je spustit řešení na osobním počítači a manuálně změřit časovou a paměťovou náročnost, ale výsledky nebudou kvalitní. Hodnotící prahy nejsou závislé pouze na úloze samotné, ale i na dalších

aspektech. Nejvíce budou ovlivněny bezesporu hardwarem. Na hardwarově odlišných výpočetních zařízeních se budou hodnoty různit. Pokud je navíc výpočetní zařízení pod zatížením (například některá běžící aplikace vykonává jakousi výpočetní činnost), spuštěné řešení získá méně procesorového času a běh úlohy může být delší.

Systém CoDiAna nabízí možnost nahrát na server vzorové řešení pedagoga. Časové a paměťové limity budou automaticky naměřeny z běhu přijatého řešení. Lze tak efektivně určit tyto hranice bez nutnosti manuálního měření. Výsledky budou automaticky doplněny do nastavení úlohy. To poskytne pedagogovi přibližné informace o časové a paměťové náročnosti. Naměřené hodnoty se budou opět vztahovat ke konkrétnímu výpočetnímu stroji jako v případě manuálního měření. Rozdíl spočívá v tom, že na tomto stroji budou poté měřeny i výsledky ostatních řešení odevzdaných studenty. Časové a paměťové hodnoty jsou teoreticky měřeny za stejných podmínek. V praxi se vždy vyskytnou nepředvídatelné anomálie, které mohou měření ovlivnit.

3.4 Aktivace a deaktivace úlohy

Pokud jsou splněny všechny náležitosti úlohy (vstupní/výstupní soubory a hodnotící kritéria), je úloha připravena na aktivaci. Každá úloha se nachází v určitém stavu a na základě tohoto stavu jsou studentům dostupné určité funkce. V neaktivovaném (výchozím) stavu student nemůže zasílat řešení na server, neboť nemusí být známé všechny skutečnosti úlohy. Aktivováním pedagog potvrdí, že úloha je připravena – jedná se o nutnou podmínku pro svolení zaslání řešení. Další nutnou podmínkou je časové vymezení úlohy pedagogem. Aktivace ani deaktivace není časové omezená, lze tedy aktivovat úlohu před jejím oficiálním začátkem, nebo deaktivovat ji v případě nenadálé chyby.

3.5 Zpracování řešení

3.5.1 Odevzdání řešení

Studenti mohou odevzdat svá řešení, pokud je úloha aktivována a pokud je úloha otevřena, tzn. čas serveru je mezi hranicemi pro oficiální začátek a konec úlohy. Odevzdání řešení vždy probíhá nahráním souboru na server. Modul

CoDiAna umožnuje odevzdání řešení ve dvou formátech. První z nich je soubor s příponou odpovídající programovacímu jazyku. To, jaké přípony, resp. jaké programovací jazyky jsou přípustné, specifikuje pedagog. Druhá možnost je nahrání archivu ZIP, který obsahuje řešení. Tento způsob umožňuje odevzdat řešení, které se skládá z více souborů, což je užitečné zejména u složitých úloh. Je důležité, aby archiv obsahoval soubory ve správné struktuře a obsahoval spouštěcí soubor s korektním názvem.

Po odeslání řešení, jsou soubory uloženy v předem definované adresářové struktuře a řešení je připraveno ke zpracování. Proces zpracování je blíže popsán v kapitole 4.4 Zpracování programovacího jazyka. Po této činnosti jsou řešiteli zobrazeny výsledky – nastavení úlohy definuje, které informace může řešitel vidět. Minimálně jsou však viditelné základní informace, jako to, jakým způsobem byl proces ukončen nebo zda je odevzdané řešení správné. Modul CoDiAna uloží po odevzdání řešení záznam do databázové tabulky `codiana_queue`, která slouží pro předávání požadavků Exekutivní aplikaci. Proces řešící odevzdání však provádí další činnosti. Probíhá kontrola, zda jsou všechna předešlá řešení od daného uživatele zpracována. Pokud nastane situace a některá řešení stále čekají na vyřízení a uživatel přesto odevzdá nové řešení, budou tato řešení anulována. Systém CoDiAna uživatele na tuto skutečnost upozorní varovnou zprávou. Z tabulky `codiana_queue` budou smazány všechny požadavky na zpracování, neboť je proces pro tyto pokusy přerušen. Těmto řešením bude v tabulce `codiana_attempt` nastavena speciální hodnota stavu indikující stav přerušení.

3.5.2 Hodnocení řešení

Po zpracování řešení, je určen finální výsledek řešení. Tento výsledek je ovlivněn hned několika faktory. Po zpracování řešení má systém CoDiAna k dispozici tři hodnoty, ze kterých je určen finální výsledek. Jsou to hodnoty o časové a paměťové náročnosti a informace o tom, zda je výstupní soubor studenta shodný (dle stanovených kritérií) s referenčním výstupním souborem úlohy. K určení finálního výsledku je použit následující vztah:

$$f(x) = o(x)\lceil t(x)\rceil\lceil m(x)\rceil\left(\frac{1}{2}t(x) + \frac{1}{2}m(x)\right), \text{ kde} \qquad (1)$$

$$o(x) = \begin{cases} 0, & chybný\ výstup \\ 1, & správný\ výstup \end{cases}, \qquad H(t) \in \langle 0,1\rangle, \qquad H(m) \in \langle 0,1\rangle$$

Funkce $o(x)$ nabývá hodnot 0, pokud je výstup chybný nebo 1, pokud je výstup klasifikován jako správný. Funkce $t(x)$ určuje výsledek časové náročnosti a funkce $m(x)$ určuje výsledek náročnosti paměťové. Pokud nejsou specifikována hodnoticí kritéria úlohy, tj. nejsou určeny hodnotící prahy, potom $t(x) = m(x) = 1$. Vztah ve výsledku popisuje průměr časové a paměťové náročnosti, pouze pokud je výstup správný za předpokladu, že žádný, ani časový, ani paměťový výsledek není nulový.

3.6 Prohlížení výsledků

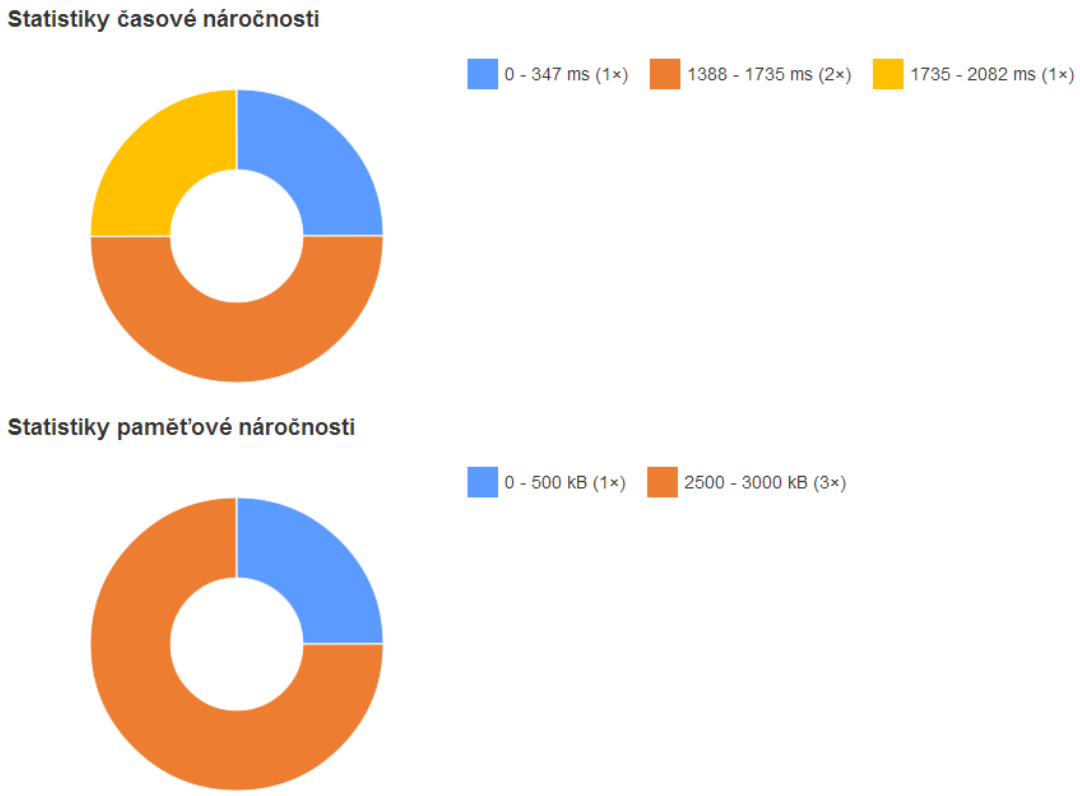
Uživatel systému Moodle může prohlížet výsledky úloh modulu CoDiAna. V závislosti na přidělených pravomocech zobrazí výsledky pro daného uživatele nebo pro všechny uživatele. Akce zobrazení výsledků pro daného uživatele se vztahuje pro studenty řešitele. Modul CoDiAna načte všechny pokusy studenta a snaží se na základě nastavení určit hodnocený pokus. Tento pokus je zobrazen jako první – jsou vyobrazeny všechny pedagogem povolené detaily pokusu. Pod tímto pokusem jsou ve zkratce vypsány výsledky ostatní pokusů. Tyto zbylé pokusy nesou méně informací, neboť nejsou systémem klasifikovány jako hodnocené řešení. Pokud má již student udělenu známku, je viditelná v přehledu úlohy.

Pedagog má oprávnění prohlížet výsledky všech uživatelů, student má tuto možnost také, ale je značně omezena. Modul CoDiAna umožňuje zobrazení detailů buď všech pokusů, nebo jen těch hodnocených. Stránka všech pokusů obsahuje tabulku, kde každý řádek reprezentuje jednotlivý pokus. Záznamy nesou všechny běžné informace, jako je jméno uživatele, ukončení, finální výsledky. Pedagog má dále možnost provádět administrativní činnost každého pokusu (například stáhnout zdrojové kódy pokusu). Zobrazení všech výsledků hodnocených pokusů je dostupné na oddělené stránce generované systémem. Pedagog má k dispozici opět všechny běžné informace jako v případě předchozí stránky v obdobné tabulce. Administrativní sekce tabulky však obsahuje více nástrojů. Je zde k dispozici stažení zdrojového kódu, úprava pokusu a detekce duplicitních řešení.

Poslední zmíněná sekce obsahuje informace o výsledcích detekce duplicitních řešení daného pokusu. Je zde zobrazena maximální nalezená podobnost daného řešení s jiným řešením. Pedagog v této sekci může spustit kontrolu daného pokusu nebo spustit jednorázově kontrolu duplicitních řešení pro celou úlohu.

Prohlížení výsledků úlohy zároveň obsahuje informace o nalezených plagiátech. Uživateli s dostatečným oprávněním jsou zobrazeny dvojice, které byly klasifikovány jako duplicitní. U každé dvojice lze vidět procentuální podobnost a je možné stáhnout zdrojové kódy pro manuální ověření.

Modul CoDiAna generuje na informativní stránce úlohy grafy, které zobrazují statistické informace o výsledcích úlohy. Jedná se o grafy pro časovou a paměťovou náročnost a výsledné stavy pokusů studentů. Grafy zobrazují data, která jsou dynamicky shlukována tak, aby nevznikalo příliš mnoho skupin. Obrázek 4 zobrazuje grafy pro testovací úlohu. Pro generování grafů byla použita Javascript knihovna `Chart.js`, vydávaná pod licencí MIT [7].



Obrázek 4: Grafy časové a paměťové náročnosti

3.7 Detekce duplicitních řešení

Modul CoDiAna umožňuje vyvolat kontrolu duplicity řešení. Kontrola je prováděna pouze na řešeních, která byla klasifikována jako hodnocená – neboť není zaručeno, že každé obdržené řešení bude vždy obsahovat validní zdrojový kód.

Pedagog má možnost vyvolat kontrolu nad celou úlohou nebo nad jednotlivým pokusem studenta. Kontrola nad celou úlohou prozkoumá všechny dvojice hodnocených řešení. Je-li n správných řešení, bude provedeno $\frac{1}{2}(n^2 - n)$ porovnání. Pokud by řešení bylo $n + 1$, je nutno provést n nových kontrol, složitost tedy stoupá exponenciálně. Porovnání dvojice proběhne pouze tehdy, pokud jsou obě řešení ve stejném programovacím jazyce a pokud Exekutivní aplikace Java podporuje detekci duplikátů nad tímto programovacích jazykem. V opačném případě není kontrola dané dvojice provedena.

Kontrola jednoho specifikovaného pokusu studenta, vyvolaná pedagogem, kontroluje daný pokus se všemi ostatními, pouze pokud jsou opět splněny výše uvedené podmínky – není nutné testovat všechny dvojice, ostatní dvojice se nijak nezmění. Vyvolání kontroly duplicitních řešení smaže všechny související výsledky předchozí kontroly. Je-li vyvolána kontrola nad řešením určitého uživatele, jsou z tabulky `codiana_plags` vymazány všechny aktuální záznamy, kde tento uživatel figuruje. Kontrola celé úlohy odstraní všechny záznamy o duplicitě, které souvisí s danou úlohou. Detekování duplicitních řešení nově vygeneruje požadované výsledky. Algoritmus porovnávání řešení je popsán v kapitole 4.5 Detekce plagiátorství programovacího jazyka.

3.8 Známkování řešení

Systém Moodle obsahuje prostředky pro udělování známek z různých aktivit. Modul CoDiAna těchto prostředků využívá. Po zpracování řešení je možné studentovi přidělit známku na stupnici od 0 do 100, kde 0 značí nejhorší možnou známku a 100 výbornou známku. Modul CoDiAna navrhne známku, která odpovídá finálnímu výsledku studenta v dané úloze. Pokud mezi řešeními studenta není žádné korektní nebo pokud žádné neodevzdal, navržená známka je 0. V opačném

případě je navržená známka vypočtena na základě finálního hodnoceného výsledku pokusu studenta, viz kapitola 3.5.2 Hodnocení řešení. Pokud je řešení studenta označeno jako plagiát, je u navržené známky varování a navržená známka je 0. Student má možnost po oznámkování shlédnout své výsledky v systému Moodle.

3.9 Pravomoci a oprávnění

Modul CoDiAna má definovanou sadu pravomocí, které jsou v modulu využívány. Pro každou akci systému je dána určitá pravomoc. Tímto je zajištěno jemné rozlišení pravomocí v modulu – přidělení oprávnění jiným uživatelům je exaktní.

Tabulka 1: Přehled pravomocí a oprávnění modulu CoDiAna

Pravomoc	Co umožňuje	Pedagog	Student
Zhlédnutí vlastních pokusů	Prohlížení výsledků vlastních pokusů.	**Zakázáno**	**Povoleno**
Odevzdání řešení	Odevzdání řešení úlohy.	**Zakázáno**	**Povoleno**
Zhlednutí některých výsledků úlohy	Prohlížení pedagogem definovaných výsledku úlohy.	**Zakázáno**	**Povoleno**
Zhlédnutí všech výsledků úlohy	Prohlížení všech výsledků dané úlohy.	**Povoleno**	**Zakázáno**
Vytvoření instance	Vytváření nových instancí modulu CoDiAna.	**Povoleno**	**Zakázáno**
Správa souborů	Nahrávání nebo generování souborů úlohy. Zahrnuje i odevzdání referenčního řešení.	**Povoleno**	**Zakázáno**
Změna stavu úlohy	Aktivování nebo deaktivování úlohy.	**Povoleno**	**Zakázáno**
Detekce duplikátů	Zkontrolování duplicity nad celou úlohou nebo nad pokusem studenta.	**Povoleno**	**Zakázáno**
Známkování	Známkování všech řešitelů úlohy.	**Povoleno**	**Zakázáno**
Úprava pokusů	Úprava pokusů studentů (přidání komentářů, změna finálního výsledku, apod.).	**Povoleno**	**Zakázáno**

Tabulka 1 ukazuje přehled všech pravomocí v modulu CoDiAna spolu s výchozím nastavením oprávnění. Lze vidět, že možnosti pedagoga a studenta jsou zcela odlišné – nemají společné žádné oprávnění. Student má možnost pouze odevzdávat řešení, vidět vlastní výsledky a vidět výsledky ostatních v omezené podobě, vidí pouze některé, pedagogem specifikované, parametry výsledků ostatních řešitelů. Možnosti pedagoga jsou daleko rozsáhlejší, spravuje veškeré nastavení úlohy. Pravomoci, které nejsou pro pedagoga povoleny, může získat jednoduše přepnutím se do role student. Pedagog tak může vidět úlohu pohledem studenta.

4 Exekutivní aplikace CoDiAna

Modul CoDiAna vyžaduje Exekutivní aplikaci reagující na zasílané požadavky. Hlavní účel Exekutivní aplikace CoDiAna je naslouchání a následné zpracování požadavku (jedná se o kompilaci, spuštění nebo detekci duplicit). Jako komunikační prostředník slouží databáze modulu CoDiAna, která obsahuje všechny potřebné informace o tom, jaká činnost má být provedena. Existují dva způsoby, kterými Exekutivní aplikace CoDiAna zjistí, že je třeba vykonávat nějakou činnost, jedná se o:

1. periodickou kontrolu databáze,
2. impuls od modulu CoDiAna.

Exekutivní aplikace CoDiAna využívá obě metody současně. Princip periodické kontroly databáze je následující: V definovaném časovém intervalu se Exekutivní aplikace CoDiAna připojí k databázi systému Moodle a nahlédne do tabulky `codiana_queue`, kde jsou uloženy všechny nevyřízené požadavky od modulu CoDiAna. Pokud není tabulka prázdná, začne proces zpracování požadavků. Tento způsob je spolehlivý, ale neefektivní. Aby byly požadavky zpracovány okamžitě, bylo by nutné kontrolovat tabulku neustále, proto byl zvolen časový interval, který je kompromisem mezi rychlostí zpracování a zátěží databáze.

Druhý způsob, který upozorní Exekutivní aplikaci CoDiAna na nový požadavek ke zpracování, je zaslání impulsu modulem CoDiAna. Pokud modul vyžaduje služby Exekutivní aplikace, vyšle jednoduchý socket obsahující klíčovou frázi. Obě strany musí mít definovány číslo portu a klíčovou frázi. Dále musí mít Exekutivní aplikace definován seznam přípustných IP adres, ze kterých může být socket zaslán a modul musí znát IP adresu, na které je spuštěna Exekutivní aplikace. Exekutivní aplikace naslouchá na definovaném portu, a pokud přijme socket, který obsahuje správnou frázi a adresa odesílatele patří mezi povolené, spustí proces zpracování požadavků. Tento způsob umožnuje okamžitou reakci aplikace na nový požadavek. Pokud není aplikace nebo modul nakonfigurován správně, nebude zpracován žádný požadavek. Z tohoto důvodu je spuštěna i periodická kontrola databáze.

4.1 Struktura aplikace

Exekutivní aplikace je programována v jazyce Java a obsahuje celkem osm hlavních balíčků, které zodpovídají za určité skutečnosti. Každý balíček obsahuje třídy, rozhraní popř. další balíčky, které pomáhají řešit určitou problematiku. Mimo základní strukturu balíčků je přítomna hlavní třída `Main`, která slouží pro spuštění aplikace. Tato třída obstarává inicializaci globálních proměnných a také spuštění vlákna pro periodickou kontrolu databáze a vlákna pro naslouchání na daném portu pro impuls od modulu CoDiAna. Po této inicializační činnosti je hlavní metoda třídy `Main` ukončena a řízení chodu aplikace je rozděleno do dvou výše zmíněných vláken.

V celé aplikaci jsou hojně využívána rozhraní definující funkcionalitu objektů. Většina tříd implementuje určité rozhraní, aby bylo možné nahradit specifickou třídu jinou třídou, která může implementovat metody odlišným způsobem, ale se stejnými výsledky. Pro tyto účely je používán návrhový vzor `Factory`, který vrací instance různých tříd maskující se za stejné rozhraní. Každá logická sekce kódu aplikace, většinou reprezentována balíčkem, obsahuje vlastní třídy výjimek. V kódu jsou pak výjimky potenciálně nebezpečných operací, které vyžadují zachycení výjimek konstrukcí `try-catch-finally`, přeposílány do vyšších úrovní algoritmu. Přeposílání zachytí všechny výjimky, které mohou nastat a jsou předány vlastní třídě výjimky v konstruktoru. Ve vyšších vrstvách algoritmu je pak nutné zachytit pouze jeden typ výjimky.

4.1.1 Externí knihovny

Exekutivní aplikace využívá celkem 3 externí knihovny. Tyto knihovny rozšiřují funkcionalitu aplikace a zjednodušují některé skutečnosti. Nezbytnou knihovnou je ovladač pro databázi MySQL, který je aplikací přímo vyžadován. Bez této knihovny by nebylo možné navázat žádné databázové spojení. Pro podporu vzdálené správy databáze je použita knihovna `Jsch`[4] [9], která zodpovídá za SSH [1] spojení a přesměrování portů. Práce s archivy, která nastává v akci detekce duplicitních řešení, je realizována prostřednictvím knihovny `zip4j` [15], pomocí které lze jednoduše číst data z archivů, resp. vytvářet archívy samotné.

[4] Java secure Channel

Další knihovny, které Exekutivní aplikace využívá, se načítají dynamicky. Jedná se o knihovny pro podporu zpracování programovacího jazyka nebo knihovny pro podporu detekce duplicitních řešení programovacího jazyka. Načítání probíhá dynamicky, jelikož předem není známo, jaké programovací jazyky budou aplikací podporované. To jaké knihovny budou načteny, je určeno v konfiguračním souboru. Více v kapitole 4.2 Modularita aplikace.

4.1.2 Jádro aplikace

Základní balíček `core` obsahuje zdrojové kódy zajišťující inicializaci aplikace. V tomto balíčku lze nalézt třídu pro načtení konfiguračního souboru. Třída obsahuje metody pro parsování XML souborů a načtení všech potřebných konstant. Pokud nastane chyba při zpracování souboru, je vyhozena výjimka s popisem chyby. Výjimka může také nastat, pokud v konfiguračním souboru nejsou všechny potřebné hodnoty. Tato třída obsahuje metody, které vrací požadované konfigurační hodnoty. Balíček `core` obsahuje třídu pro globální vytvoření a uchování instancí tříd – jedná se především o třídy, které využívají návrhový vzor `Singleton`. Příkladem může být například třída pro databázové spojení nebo třída konfiguračního souboru. Kdekoli v programu je tedy přístupný tento globální objekt, který obsahuje reference na důležité komponenty aplikace.

4.1.3 Databázové spojení

Komunikační balíček, který zajišťuje všechny záležitosti ohledně databáze, nese název `database`. Obsahuje základní rozhraní pro objekty spravující databázi systému Moodle. Jsou zde definované hlavičky různých metod, které slouží pro načítání dat, aktualizaci a mazání záznamů nebo vložení nových záznamů. Každá tabulka modulu CoDiAna je reprezentována třídou, která obsahuje všechny údaje o záznamu. Pokud tyto objekty zrcadlí tabulku, ve který je definován referenční odkaz (například `id` úlohy), umožňuje objekt načtení tohoto objektu dynamicky pomocí definovaných metod. Výsledné třídy reprezentující tabulky jsou `Úloha`, `Požadavek`, `Pokus`, `Uživatel` a `Plagiát`. Vazby mezi objekty mohou být například následující: Každá instance objektu `Pokus` může dynamicky načíst záznamy z požadovaných tabulek a získat referenci na instanci objektu `Uživatel`

nebo instanci objektu `Úloha`. Každá z těchto tříd implementuje rozhraní, které definuje základní vlastnosti daných objektů.

Třída zodpovídající za správu lokální databáze obsahuje SQL příkazy, které vykonávají žádané operace. Aby mohla být provedena určitá operace, je nutné úspěšné připojení do databáze. Poté třída vytvoří instanci objektu `Statement`, který slouží pro přípravu daného příkazu. Tato instance přebírá řetězec obsahující SQL příkaz, ve kterém však nejsou definovány žádné specifické hodnoty. Hodnoty jsou definovány později pomocí metod, které zajistí bezpečné přijetí hodnot.

Následující příklad demonstruje vymazání záznamu z databáze. Je definován SQL příkaz pro smazání záznamu z tabulky `codiana_queue` na základě unikátního identifikátoru `id`. Příkaz vypadá následovně:

```
DELETE FROM  ::codiana_queue
             WHERE ( id = ? )
LIMIT 1;
```

Použití příkazu ve třídě pro správu databáze systému Moodle pak vytvoří instanci objektu `Statement` a na základě předané instance objektu `Požadavek`, který byl načten v jiné části programu, je doplněn unikátní identifikátor instance objektu. Výsledek vypadá následovně:

```
statement = createPreparedStatement (DELETE_QUEUE_ITEM);
statement.setInt (1, item.getId ());
statement.executeUpdate ();
```

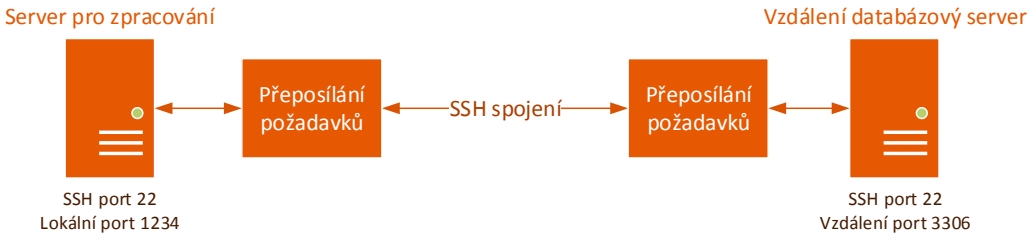
Obdobným způsobem jsou realizovány ostatní metody. U některých operací není znám výsledný příkaz. Například u operace vložení záznamů reprezentující duplicitní řešení není znám přesný počet nalezených dvojic plagiátů, dokud není provedena kontrola. Problematiku je možné realizovat provedením jednoduché operace vložením jednoho záznamu n-krát, kde n je počet nalezených řešení. Toto řešení však není optimální, neboť databáze musí zpracovat a provést n příkazů. Třída pro správu databáze dynamicky vytvoří výsledný příkaz a nastaví požadované argumenty příkazu. Databáze tak musí zpracovat pouze jeden delší

příkaz což je mnohem efektivnější. Režie pro zpracování příkazu pro databázi MySQL je dána různými faktory [12].

Ty jsou definované následovně:

- připojení k databázi (3 body)
- odeslání příkazu (2 body)
- parsování příkazu (2 body)
- vložení jednoho záznamu (1 bod)
- ukončení (1 bod)

Balíček `database` také obsahuje třídu pro správu vzdálené databáze. Tato třída opět implementuje rozhraní pro správu databáze, v kódu tak může být použita jak třída pro vzdálenou i lokální správu databáze. Třída rozšiřuje třídu pro správu lokální databáze a přepisuje pouze inicializační metodu. V případě lokální implementace zde probíhá připojení k databázi a nastavení kódování. Třída vzdálené správy databáze nejprve vytvoří SSH připojení na daný server za pomoci údajů v konfiguračním souboru. Po úspěšném připojení je vytvořen tunel pro přesměrování portů. Třída zasílá požadavky na daný port jako v předchozím případě, s tím rozdílem, že všechny požadavky jsou přeposílány na jiný server a definovaný port. Následující obrázek demonstruje jednoduchý SSH tunel, který přeposílá požadavky z portu 1234 na vzdálený server pomocí SSH spojení na port 3306 [1].



Obrázek 5: Přeposílání požadavků přes SSH spojení

4.1.4 Kontrolní balíček

Je-li úspěšně navázáno databázové spojení, je možná kontrola tabulky `codiana_queue` pro případné požadavky. Základní balíček `controls` obsahuje třídy a rozhraní, které definují, jak má vypadat kontrola databáze a jakým způsobem jsou zpracovány požadavky. Rozhraní pouze definuje povinnou metodu `update`, která zpracovává výše uvedené požadavky. Třída implementující toto rozhraní obsahuje algoritmus pro korektní globální kontrolu databáze a následné zpracování požadavků. Celý proces řídí třída `Controller` z balíčku `controls`. Po načtení všech požadavků z tabulky `codiana_queue`, jsou jednotlivě zpracovány. Pořadí zpracování požadavků je určeno prioritou záznamu ve frontě a časem vytvoření požadavku. Vyřízení požadavku se skládá z několika kroků a různí se v závislosti na typu požadavku. Požadavky typu zpracování řešení a měření referenčních hodnot úlohy jsou zpracovány následujícím způsobem:

1) Načtení souvisejících záznamů z databáze.
   - V této části algoritmu je načten záznam pokusu řešitele, který obsahuje všechny potřebné detaily o pokusu (programovací jazyk, čas odeslání, číslo studenta apod.).
2) Na základě programovacího jazyka řešení je vytvořena instance třídy pro zpracování programovacího jazyka.
   - Návrhovým vzorem `Factory` je zažádáno o instanci. Pokud programovací jazyk není podporovaný, je zpracování ukončeno.
3) V závislosti na charakteru programovacího jazyka je provedena kompilace jazyka.
   - Pokud je jazyk kompilovatelný, jsou vytvořeny konfigurační přepravky[5] a je zahájen proces kompilace. Tento proces se skládá ze tří částí (příprava, získání argumentů příkazu a následný „úklid“ po kompilaci).
   - Balíček pro zpracování zkompiluje řešení a vytvoří přepravku, která obsahuje informace o běhu procesu.

[5] Jednoduchá třída, která slouží pro zapouzdření a přepravu dat

- Pokud byl proces kompilace neúspěšný, zpracování je přerušeno a výsledek je uložen do databáze.

4) Spuštění řešení
   - Pro tuto činnost je opět využit balíček pro zpracování, který vytváří přepravku s informacemi o běhu procesu. Při neúspěchu spuštění jsou do databáze uloženy informace o chybě a je nastaven příslušný stav pokusu (časový limit, chyba při spuštění, apod.).
5) Ohodnocení nebo naměření hodnot běhu pokusu
   - V závislosti na typu požadavku je provedeno buď ohodnocení řešení, nebo měření referenčních hodnot úlohy. Po této činnosti jsou opět uloženy detaily do databáze. V případě měření hodnot je upraven záznam v tabulce `codiana` (jsou editovány hodnoty času a paměti pro hodnocení pokusů).

Systém CoDiAna umožňuje detekci kontroly duplicity pro všechna hodnocená řešení úlohy nebo jen pro jeden vybraný hodnocený pokus řešitele. Algoritmus pro tyto typy požadavků je následující:

1) Načtení souvisejících záznamů.
   - Na základě unikátního identifikátoru úlohy jsou načteny všechny hodnocené pokusy řešitelů v úloze a uloženy do pole.
2) Seřazení pole
   - Pokud se jedná o kontrolu jednoho řešení, je pole seřazeno tak, aby na začátku byla instance třídy `Pokus`, která má být kontrolována.
3) Určení podobnosti všech dvojic.
   - V závislosti na typu požadavku je určena podobnost mezi všemi dvojicemi nebo mezi dvojicemi, které obsahují žádaný pokus. Pokud je kontrolováno pouze jedno řešení, algoritmus se ukončí dříve, neboť je proveden pouze jeden krok vnějšího cyklu. Proto je vyžadováno, aby první prvek pole byl hledaný hodnocený pokus). Zjednodušený pseudokód pro algoritmus, který realizuje, jakým způsobem jsou vybírány dvojice pro porovnávání, je popsán dále.

```
plagiaty = Null
pamet = Null
For (i = 0; i < n; i++) {
      pamet[i] = priprav_reseni (reseni[i])
}

For (i = 0; i < n-1; i++) {
      plugin_pro_jazyk = Factory.ziskej_instanci (reseni[i])

      For (j = i + 1; j < n; j++) {

           If  priprava_selhala (pamet[i]) nebo
               priprava_selhala (pamet[j])
               neni plugin_pro_jazyk
                     preskoc

           vysledek = plugin_pro_jazyk.porovnej(pamet[i], pamet[j])

           If vysledek.podobnost > prah_podobnosti
                plagiaty[] = vysledek
      }

      If kontrola_pouze_jednoho_reseni
           ukonci
}
```

4) Filtrování výsledků.
   - Pokud je podobnost vyšší než specifikovaný práh, je dvojice označena jako duplicitní a přidána do seznamu nalezených duplicitních řešení.
5) Uložení výsledků.
   - Všechna nalezená duplicitní řešení jsou jednorázově uložena do tabulky `codiana_plags`. Označené dvojice obsahují údaje o kontrole plagiátorství (identifikační čísla obou uživatelů, procentuální pravděpodobnost).

Pokud dojde k nenadálé chybě při zpracování, snaží se Exekutivní aplikace uložit informace o chybě do databáze. Po zpracování požadavku (ať už úspěšném nebo neúspěšném) je zpracován další požadavek. Pokud byly zpracovány všechny požadavky, zpracování je ukončeno. Aplikace vyčkává na impuls od modulu CoDiAna nebo na uplynutí doby periodické aktualizace.

4.1.5 Balíček zpracování řešení

Jednou z hlavních činností Exekutivní aplikace je zpracování řešení. Kód pro tyto činnosti je obsažen v balíčku `processing`. Tento balíček obsahuje více rozhraní než skutečných tříd, jelikož Exekutivní aplikace sama o sobě nedokáže zpracovat žádný programovací jazyk. Pouze definuje rozhraní, která jsou využívána pluginy pro tuto činnost. Balíček je rozdělen na tři základní logické sekce. První sekce specifikuje požadavky pluginů pro programovací jazyky, které jsou spustitelné[6]. Jsou zde definovány nejenom metody pro zpracováním řešení, ale i jiné třídy, které se zpracováním souvisí. Jsou to přepravky, které slouží pouze pro ukládání výsledků nebo také výjimky, které souvisí s chybou při spuštění daného řešení. Důležitou přepravkou je konfigurační přepravka. Tato třída obsahuje různá nastavení a informace o řešení (například, kde jsou uloženy soubory řešení apod.) Obdobně jako pro spustitelné programovací jazyky je definována sekce obsahující rozhraní, přepravky a výjimky pro programovací jazyky, které jsou kompilovatelné. Vztah mezi spustitelnými a kompilovatelnými jazyky je realizován za pomoci OOP (zejména dědičnosti). Třetí logická sekce provádí spouštění příkazů pro zpracování řešení. Jedná se o třídu, která realizuje přípravu, spuštění a zachycení výsledků příkazu. Třída již nevyhodnocuje, zda byl výsledek správný nebo špatný, to je už v kompetenci jiných tříd, které se zabývají vyhodnocením výsledku.

4.1.6 Hodnocení výsledků

To, jakým způsobem jsou řešení hodnocena, závisí na několika aspektech. Jedná se o nastavení úlohy a výsledky spuštění řešení. Aby mohlo být řešení ohodnoceno, je nutné, aby při spuštění (popř. kompilaci) nenastaly žádné chyby a byl k dispozici validní výsledek naměřených hodnot a vygenerovaný výstupní soubor. Bez těchto náležitostí není možné ohodnotit řešení jinak než známkou reprezentující neúspěch, procentuálně tedy hodnotou 0 %. Při spuštění je vygenerován objekt, který obsahuje potřebné podrobnosti o běhu procesu, jmenovitě délka běhu procesu, průměrná paměťová náročnost, kód ukončení procesu a počet řádků výstupního souboru. Tento objekt je klíčový k určení

[6] Označení pro skupinu jazyků, u kterých není nutná příprava pro spuštění

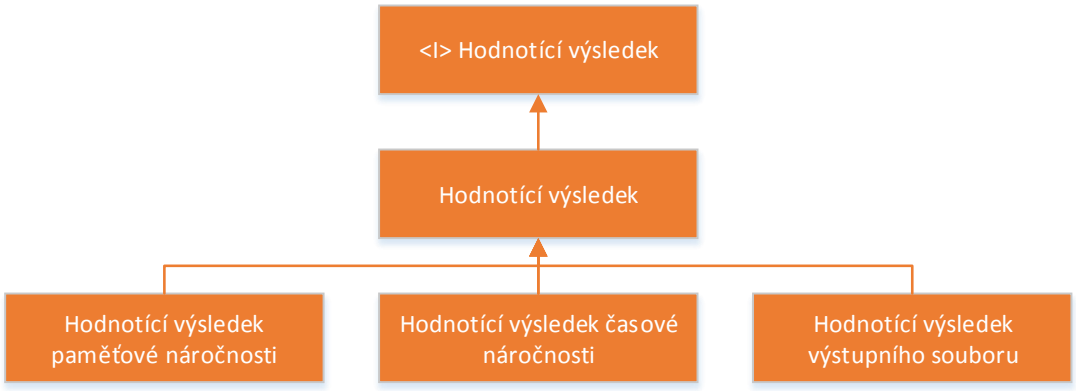
hodnocení řešení. Nastavení úlohy, zejména hodnotící prahy pro paměťovou a časovou náročnost, ovlivňuje výsledky hodnocení. Výsledek může být také ovlivněn módem porovnávání výstupních souborů. Návrh Exekutivní aplikace umožňuje specifikovat módy nejen při porovnávání výstupů, ale i při určení výsledků z paměťových a časových naměřených hodnot. Výchozí hodnota tohoto módu je značena jako prahová, neboť jsou specifikovány hranice, na základě kterých je určen výsledek.

Celý proces známkování začíná porovnáním výstupních souborů. Jsou postupně načítány oba výstupní soubory (referenční výstupní soubor a výstupní soubor právě hodnoceného řešení). Na základě módu porovnávání výstupních souborů je určena logická hodnota, která definuje, zda jsou výstupní soubory obsahově stejné. Pokud nabývá výsledek hodnoty nepravda, hodnocení je ukončeno a danému pokusu aplikace přiřadí známku 0 %, stav pokusu se změní na hodnotu `nesprávná odpověď`.

Za podmínky validního výstupního souboru je přistoupeno ke kontrole dalších detailů běhu řešení. První je kontrola časové náročnosti. Aplikace načte z tabulky `codiana` současné hodnoty hodnotících prahů pro časovou náročnost a v závislosti na hodnotách prahů a naměřené časové náročnosti určí procentuální výsledek. Při překročení maximální povolené časové náročnosti aplikace přeruší hodnocení, udělí známku 0 % a nastaví stav pokusu na `překročen časový limit`.

Poslední hodnotící veličinou je paměťová náročnost, ta je hodnocena obdobně jako časová – opět tedy s pomocí hodnotících prahů úlohy. Pokud má paměťová náročnost vyšší než povolenou hodnotu je pokus hodnocen známkou 0 % a nastaven stav pokusu na hodnotu `překročen paměťový limit`. Pokud jsou naměřeny jiné než standardní veličiny (časová, paměťová náročnost a výstupní soubor), mohou být ohodnoceny v této části algoritmu. Určení známky, jiné než 0 %, vyžaduje, aby byly všechny předchozí výsledky nenulové. Poté je výsledná známka průměrem procentuálních hodnot časového a paměťového výsledku (resp. průměrem všech ostatních hodnocení). Hodnocení je určeno Exekutivní aplikací, může být však manuálně upraveno pedagogem nebo pověřeným uživatelem.

Balíček `grading` obsahuje rozhraní pro známkovací moduly. Každé rozhraní definuje známkovací metodu, která vrací hodnotící výsledek. Tento výsledek není pouze číslo, ale jedná se o rozhraní definující metodu pro vrácení výsledku hodnocení. Každá hodnotící položka může využít této skutečnosti a předávat v hodnotícím výsledku i jiné informace než procentuální výsledek. Každá třída implementující rozhraní pro známkovací modul musí dále implementovat metodu pro nastavení konfigurace, která mimo jiné obsahuje potřebné hodnoty ke známkování (například hodnotící prahy a objekt s naměřenými hodnotami). Pro každou hodnotící položku je vytvořeno rozhraní, které definuje přijímací parametry potřebné k hodnocení. Za pomocí návrhového vzoru `Factory` jsou vytvořeny tovární třídy. Stejně jako má hodnotící výsledek společného předka, mají jej i všechny módy známkovacích modulů. Příkladem může být určení výsledku výstupního souboru. Je vytvořena abstraktní třída pro výstupní známkovací modul rozšiřující abstraktní třídu známkovacího modulu, který implementuje rozhraní známkovacího modulu. Abstraktní třídu výstupního známkovacího modulu rozšiřují tři známkovací třídy reprezentující jednotlivé módy porovnávaní výstupních souborů. Následující schéma zobrazuje strukturu hodnotících výsledků. Stejně jako je definována struktura hodnotících výsledků, je definována struktura pro nastavení konfigurace známkovacích modulů.



Obrázek 6: Struktura hodnocených výsledků

Ohodnocení řešení probíhá tak, že je vytvořen objekt třídy `Monitor`, který obsahuje všechny moduly hodnocení, které budou nad řešením spuštěny. Tento způsob umožňuje přidání dalších hodnotících parametrů, které mohou být monitorovány po rozšíření systému. Objekt vlastnící reference monitorů zajišťuje, aby nemohly být přidány monitory stejného typu. Zároveň vyžaduje alespoň jednu referenci každého ze základních typu monitoru, aby mohl být proveden hodnotící algoritmus. Po přidání všech monitorů je jednorázově spuštěno hodnocení a jsou vyhodnoceny výsledky.

4.1.7 Balíček detekce duplicitních řešení

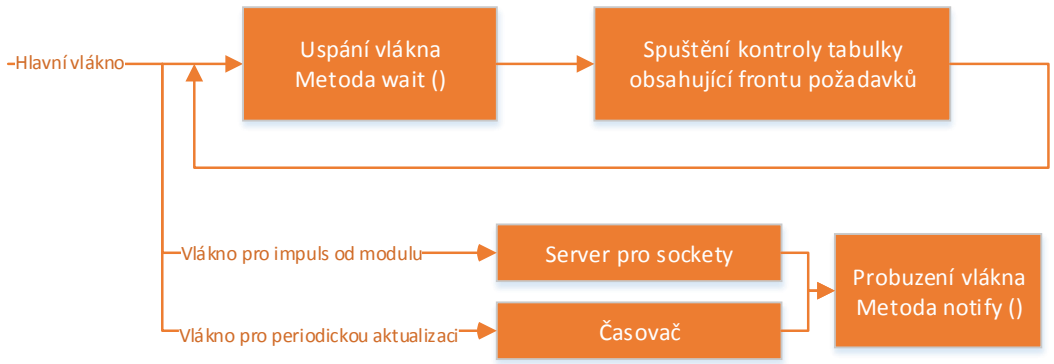
Struktura balíčku `plagiarism` slouží pro detekci duplicitních řešení a stejně jako v případě balíčku `processing`, jsou zde definovaná převážně rozhraní. Základní aplikace nepodporuje detekci duplicitních řešení žádného programovacího jazyka. V balíčku `plagiarism` jsou specifikované metody a přepravky, které musí implementovat každý plugin, který chce provádět tuto činnost. Hlavní je rozhraní označené jako `ILanguageComparator`, které obsahuje tři základní metody. Metody vesměs přejímají parametry typu `File` reprezentující soubor nebo adresář. Řešení může obsahovat více souborů, je nutné předávat kořenové adresáře, ve kterých jsou uloženy všechny soubory.

První metoda rozhraní přejímá dva parametry, adresáře, kde jsou uloženy všechny soubory porovnávaných řešení. Metoda porovná dvě zadaná řešení a vrátí výsledek. Návratový typ je přepravka, která obsahuje hodnotu naměřené rozdílnosti a maximální rozdílnosti zkoumaných objektů, ze které je možné lehce určit pravděpodobnost objektů. Tato metoda vždy načte všechny soubory daného adresáře a následně je zpracuje a provede porovnání řešení. Druhá metoda slouží pro přípravu řešení. Metodě je předán adresář, ve kterém jsou uloženy soubory řešení a třída musí soubory načíst a připravit pro porovnání. S touto metodou úzce souvisí poslední metoda, která slouží pro smazání dočasných předpočítaných výsledků.

V balíčku `plagiarism` lze nalézt třídy výjimek, které jsou využívány v případě chyby při zpracování nebo porovnávání řešení. Dále jsou zde přítomny přepravky, které obsahují dvojici porovnaných řešení nebo také přepravku, která obsahuje všechna načtená řešení dané úlohy.

4.1.8 Detekce impulsu modulu CoDiAna

Exekutivní aplikace může spustit kontrolu tabulky `codiana_queue` periodicky nebo impulsem, které vyšle modul CoDiAna. Balíček `net` obsahuje celkem dvě třídy realizující tuto činnost. První je třída slouží pro přijímání socketů na daném portu. Konfigurační soubor Exekutivní aplikace obsahuje číslo portu, na kterém bude socket server naslouchat a klíčovou frázi, pomocí které se příchozí socket autentizuje. Třída implementuje rozhraní `Runnable` a v metodě `run` čeká na příchozí paket v nekonečné smyčce. Pokud na definovaný port dorazí socket, je vytvořena přepravka, která obsahuje informace o validitě přijatého socketu. Jsou zde uloženy detaily o adrese socketu a přijaté zprávě socketu. Pokud je socket označen jako validní, je probuzeno hlavní vlákno, které čeká na tento impuls. Následující ilustrace zobrazuje celý proces.



Obrázek 7: Struktura vláken v Exekutivní aplikaci

4.1.9 Balíčky pro logování a utility

Proces logování je pro vývojáře velice důležitý. Pokud nastane při běhu programu neznámá chyba, je velice důležité uložit co nejvíce informací o této chybě (aby mohla být lokalizována a opravena). Logování může sloužit i jako nástroj pro ladění, a to zejména při asynchronních událostech. Exekutivní aplikace obsahuje třídu, která slouží pro zaznamenávání požadovaných informací do

souborů. Tato třída využívá zabudované třídy pro logování a definuje různé podoby volání logovacího procesu. Tato třída zároveň aktivuje zápis do souboru. Veškeré logované údaje jsou postupně připisovány do definovaného souboru. Tento soubor je klíčový pro odhalování chyb v případě havárie systému. Všechny související třídy s logovacím procesem jsou uloženy v balíčku `logging`.

Poslední balíček aplikace nese název `utils` a obsahuje podpůrné prostředky. Jsou zde uloženy například třídy pro práci s řetězci nebo se soubory. Balíček dále obsahuje třídu `DataProvider`, která zapouzdřuje proces parsování záznamů z databáze a zároveň ukládá objekty do paměti a vytváří tak primitivní `cache`[7]. Pokud je pak v programu žádost o získání objektu z databáze (například objekt `Uživatel`) tato třída na základě unikátního identifikátoru vrátí uloženou instanci v paměti, pokud už byla načtena, nebo zažádá o její načtení a uloží výsledek do paměti pro budoucí volání. Aby se předešlo nekonzistenci dat, je `cache` při zpracování každého požadavku modulem CoDiAna vyprázdněna.

4.2 Modularita aplikace

Aplikace je navržena tak, aby byla umožněna podpora libovolných programovacích jazyků. Exekutivní aplikace dynamicky načítá knihovny, které doplňují žádanou funkcionalitu. Každá knihovna, tj. soubor typu `jar`, obsahuje nejméně jednu třídu (modul), která implementuje určité metody. V základní podobě (bez knihoven) nepodporuje aplikace žádný jazyk, ani nedokáže odhalit žádné duplikáty. Aplikace je zodpovědná za následující:

- načítání externích knihoven,
- práce s databází (načtení, smazání, úprava a vložení),
- volaní metod modulů a následná kontrola jejich běhu,
- spouštění skriptu, který vykoná určitý příkaz (viz kapitola 4.4 Zpracování programovacího jazyka),
- odchytávání a zpracování výjimek metod modulů.

Exekutivní aplikace obsahuje algoritmy, které umožňují zpracování požadavků od modulu CoDiAna – požadavky mohou být: detekce duplikátů, zpracování řešení

[7] dočasná paměť

a měření hodnot pro určení časové a paměťové náročnosti. V algoritmu je využíváno několik návrhových vzorů, nejdůležitější pro zajištění modularity aplikace je návrhový vzor `Factory`, který umožňuje vytváření různých objektů v průběhu programu na základě určité aktuální informace. To je velice užitečné, neboť lze dynamicky načíst potřebný programovací modul, který dokáže zpracovat daný programovací jazyk.

Aby mohly být moduly načítány s využitím návrhového vzoru `Factory`, je nutné mít definované rozhraní, za které bude vytvořený objekt považován. Exekutivní aplikace definuje několik rozhraní, počínaje klíčovou částí modulů, konče definovanými výjimkami a přepravkami. Exekutivní aplikace podporuje dva druhy modulů – podporu zpracování daného jazyka a podporu detekce plagiátorství. Je tedy možné, aby systém CoDiAna podporoval určitý jazyk jenom z části – dokáže jej pouze zpracovat, nedokáže však detekovat duplicitní kódy. Důvodů je několik. Problémy zpracování a detekce jsou velice rozlišné. Přidání podpory zpracování jazyka je vesměs dodefinovaní příkazů pro spuštění kódu. Detekce duplicity je složitější problém, neboť je nutné porozumět jazyku samotnému. Je nutné znát jeho syntaxi, strukturu a možnosti programovacího jazyka.

4.3 Podpora více programovacích jazyků

Systém CoDiAna umožňuje řešení úlohy různými programovacími jazyky. Pedagog nebo pověřený uživatel může tuto skutečnost měnit. Řešitel má tak možnost, za předpokladu, že je povoleno více jazyků u jedné úlohy, odeslat řešení v různých jazycích. Tato skutečnost přináší řadu dalších aspektů. Každý programovací jazyk zpracovává instrukce jinou rychlostí. Některé programy jsou v některých jazycích rychlejší než jiné. Řešitel má možnost zaslat kód v jiném programovacím jazyce, pokud jeho dosavadní řešení v původním jazyce nesplňuje minimální časové kritérium nebo pokud je výsledek pro řešitele neuspokojivý. Analogicky lze provést stejné opatření pro paměťovou náročnost. Podpora více jazyků dává navíc řešitelům prostor programovat v jazyce, který řešitel preferuje.

4.4 Zpracování programovacího jazyka

Je nezbytné, aby byla Exekutivní aplikace spuštěna na unixovém operačním systému z důvodu bezpečnosti a práce s procesy. Na cílovém operačním systému byl vytvořen skript, který spustí definovaný proces. Skript však provádí další činnosti. Vhodnými parametry přesměrovává proudy dat procesu. Při běhu zároveň monitoruje náročnost procesu (paměťovou a časovou) a zodpovídá za ukončení procesu po určité době (pokud proces běží delší než maximální definovanou dobu nebo pokud proces překročí paměťové prostředky). Skript dokáže podat informace o běhu procesu (včetně jeho ukončení). Důležitou vlastností skriptu jsou dále bezpečnostní opatření, která omezují spouštěné procesy. Volání skriptu je následující:

```
./exec.sh    –i vstup –o vystup –e chybovy_vystup
             -m pametove_omezeni –t casove_omezeni
             -c prikaz argument1 argument2
```

Spouštěcí skript vyžaduje příkaz, který bude vykonán. Vše, co se nachází po parametru `-c`, bude součástí příkazu, který bude spuštěn v kontextu jiného uživatele, který má omezené prostředky. Některé argumenty skriptu jsou nepovinné – pokud nejsou definovány, obsahují standardní hodnotu. Uvažujme následující skript, kde definujeme časové omezení a příkaz.

```
./exec.sh    -t 5 –c sleep 10
```

Bude vykonán příkaz, který přepne současného uživatele na jiného uživatele a spustí příkaz, který byl definován.

```
sudo         –u uzivatel –c sleep 10
```

Na operačním systému se však musí nacházet uživatel, který má omezené pravomoci. Ukázka volání skriptu spustí proces `sleep` s trváním 10 sekund. Parametr `-t` však definoval maximální dobu běhu 5 sekund – výsledkem bude přerušení spuštěného procesu `sleep`. Exekutivní aplikace využívá tento skript při zpracování všech řešení [4].

Modul pro podporu daného jazyka může být dvojí. Některé programovací jazyky vyžadují kompilaci před tím, než mohou být spuštěny. Jsou proto dvě základní rozhraní pro kompilované jazyky a nekompilované jazyky. Modul pro nekompilovaný jazyk implementuje následující:

- předání objektu s nastavení
- příprava pro spuštění,
- dodefinování parametrů pro spuštění,
- obslužná rutina po spuštění.

Rozhraní pro kompilované rozšiřuje rozhraní pro nekompilované jazyky. Musí umět výše uvedené skutečnosti. Obdobně je nutné implementovat metody, které jsou však velice podobné těm předchozím. Jsou to:

- příprava pro kompilaci,
- dodefinování parametrů pro kompilaci,
- obslužná rutina po kompilaci.

Princip je pro obě rozhraní vesměs stejný. Postup se skládá ze tří kroků. První krok je příprava – je nutné se na akci připravit. Příprava může například kontrolovat, zda všechny žádané soubory existují. Pokud řešení obsahovalo více souborů, může se jednat o nalezení spouštěcího souboru.

Druhý krok je akce samotná (spuštění nebo kompilace). Po úspěšné přípravě je nutné dodefinovat argumenty, které budou předány spouštěcímu skriptu, a spuštění žádané akce. Modul samotný by měl pouze dodefinovat parametry. O spuštění skriptu se opět postará hlavní Exekutivní aplikace, která převezme argumenty od modulu. Na závěr je provedena obslužná rutina, která by měla uvést server do stavu před kompilací/spuštěním (smazání dočasných souborů, apod.).

4.4.1 Výsledné stavy

Zpracování algoritmu může být ukončeno různými způsoby. Je brán zřetel na kompilaci, naměřené hodnoty a na další aspekty. Systém CoDiAna definuje celkem 16 různých výsledných stavů zpracování řešení. Všechny stavy však nejsou přístupné pro řešitele, některé stavy slouží pouze pro interní záležitosti. Každý stav má přiřazené unikátní číslo. Statusy jsou rozděleny do logických skupin.

První skupina je `kompilační`, která obsahuje stavy pro `chybu`, `timeout` a `úspěšnou kompilaci`. Při ukončení kompilace `chybou` je k dispozici i její popis, například `nenalezena hlavní třída` apod. Neobvyklé ukončení `timeout` je pouze pro extrémní případy, ve kterých může dojít k chybě v kompilaci, po které se proces kompilace řádně neukončí. Stav `správné kompilace` má pouze interní kontrolní využití.

Podobně jako je definována první skupina je definována druhá skupina statusů `exekuční` se stavy pro `chybu`, `timeout`, `memoryout` a `přirozené ukončení` programu. Ukončení `chybou` znamená, že program neskončil standardně, například nastala chyba v programu, která způsobila nečekané ukončení aplikace. Stav ukončení `timeout` značí, že aplikace nedokončila svou činnost v daném časovém limitu. Stav ukončení `memoryout` má obdobný účel jako `timeout` s tím rozdílem, že se jedná o paměťovou náročnost. Přirozené ukončení značí, že program skončím kódem 0 a systém CoDiAna nemusel zasahovat do běhu aplikace.

Další skupina definuje problematiku zdrojových kódů. Stavy jsou pro `validní`, `nevalidní` a `nebezpečné` zdrojové kódy. Tato skupina prozatím nemá využití neboť systém CoDiAna doposud neomezuje zdrojové kódy (například zákazem použití některých balíčků). Tyto stavy byly definovány pro případné rozšíření systému.

Čtvrtou skupinu tvoří stavy pro zvláštní případy. Stav `čekání na zpracování` je použit bezprostředně po odevzdání řešení. Spolu s ním úzce souvisí stav `zpracování přerušeno`. Stav nastane, pokud řešitel pošle řešení úlohy, ale předchozí pokusy doposud nebyly zpracovány, jsou ve stavu `čekání na zpracování`. Systém CoDiAna přeruší kontrolu těchto předešlých pokusů a nastaví jim stav `zpracování přerušeno`. Poslední status ve skupině indikuje `nespecifikovanou chybu`. Všechny ostatní chyby, které mohou nastat, jsou zaobaleny právě v tomto stavu. Například pokud nastane problém s databází systému a data nejsou korektní, může nastat tento status.

Poslední skupina specifikuje výsledky měření hodnot. Pokud je při porovnávání výstupních souborů nesrovnalost, pokus je ukončen hodnotou

`chybný výstup`. Překročení časového limitu nastaví hodnotu stavu daného pokusu na `časový limit`. Analogicky je použit stav `paměťový limit`. Za předpokladu, že jsou všechna měření úspěšná, je pokusu udělen status `měření v pořádku`. Tento kód značí správný pokus, který může být hodnocen. Tento stav je žádoucí.

4.4.2 Plugin pro zpracování programovacího jazyka Java

Součástí systému CoDiAna je i demonstrativní plugin pro podporu zpracování jazyku Java. Výsledný plugin musí implementovat kompilační metody, jelikož jazyk Java musí být před spuštěním zkompilován do bajtkódu[8]. Systém CoDiAna umožňuje, aby řešení obsahovala více souborů. Plugin pro jazyk Java se před kompilací snaží nalézt správný soubor (spustitelný). Obdobná rutina se provádí před spuštěním, kde je hledána hlavní spouštěcí třída, resp. její balíček a název. Plugin pro podporu zpracování definuje sadu příkazů pro kompilaci a spuštění.

Aby mohl být algoritmus zpracován korektně, je nutné zaslat soubor ve správné adresářové struktuře. Plugin se snaží pozměnit hlavičky přijatých souborů, pokud jsou nesprávně definované a umožnit tak zpracování přijatých zdrojových kódů. Tato oprava probíhá u řešení obsahující jediný soubor, ve kterém je definován balíček (`package`). Definice balíčku v souboru musí odrážet adresářovou strukturu souboru. Korektní způsob pro zaslání jediného souboru s definovaných balíčkem, je zaslání ZIP archivu se správnou adresářovou strukturou.

Student má také možnost odeslat řešení, které obsahuje více souborů. V tomto případě musí zaslat jeden archiv a příponou zip, který obsahuje všechny potřebné soubory. Každý soubor v archivu má pak cestu definovanou relativně vůči kořenu archivu. Ukázkou může být soubor v programovacím jazyce Java. Například pokud archiv obsahuje soubor `Sockets.java` v následující struktuře.

```
src/cz/tul/dpg/Sockets.java
```

Balíček souboru v takovéto struktuře musí reflektovat jeho relativní pozici v archivu. Musí být deklarován jako `src.cz.tul.dpg`. Je nutno brát v potaz, že

[8] Přenositelný kód

struktura v archivu může v některých programovacích jazycích ovlivňovat výsledné řešení. Pokud by řešitel použil jinou cestu balíčku než výše definovanou, jeho řešení by skončilo `kompilační chybou` a pokus by byl hodnocený jako neúspěšný.

4.4.3 Módy porovnávání výstupu

Při porovnávání výstupních souborů má pedagog možnost specifikovat způsob, jakým budou soubory porovnávány. Systém CoDiAna prozatím obsahuje tři způsoby porovnávání. Tyto způsoby vesměs určují, jak přesný musí výstup být. První metoda nazvaná `strict`, vyžaduje, aby se výstupní soubory shodovaly znak po znaku. Při porovnávání se nerozlišuje charakter znaku. Nezáleží, zda se jedná o mezeru nebo číslo. Druhý způsob porovnávání nese název `tolerant`, kde je vyžadována shoda po řádcích, s tím rozdílem, že prázdné řádky jsou ignorovány. Tento způsob je výchozí volbou při vytváření nové úlohy. Poslední způsob s názvem `vague`, vyžaduje, aby byla shodná neprázdná data. Výstupní soubory jsou načítány po tokenech, tj. sekvencích neprázdných znaků, a ty jsou poté porovnávány. Ve výsledku jsou porovnávány pouze neprázdné sekvence znaků. Všechny zmíněné metody vyžadují shodu výsledků, určují tedy, do jaké míry musí řešitel dodržet formát výsledných dat. Prázdné znaky na konci výstupních souborů jsou ignorovány ve všech třech případech.

4.4.4 Módy výběru hodnocených řešení

Řešení, která skončila úspěšně, mohou být vybrána jako hodnocená. Pokud řešitel zašle více řešení a všechna jsou klasifikována jako úspěšná, je nutno určit, které řešení bude vybráno jako hodnocené. Hodnocené řešení poté definuje výslednou známku studenta pro danou úlohu. To, jaké řešení bude vybráno, může ovlivnit pedagog nastavením úlohy. Jsou celkem tři módy výběru hodnocených řešení. První režim vybere první správné řešení. Pokud tedy student zašle po prvním správném řešení jiné, hodnocené řešení se nezmění. Druhý režim vybírá poslední správné řešení – vždy poslední správné řešení bude vybráno jako hodnocené. Poslední režim vybere řešení, které má nejvyšší finální výsledek (nejlepší časovou a paměťovou náročnost). Výchozí režim výběru je poslední správné řešení.

4.5 Detekce plagiátorství programovacího jazyka

Cílem této činnosti, je zjistit, zda při porovnávání dvou algoritmů lze říci, že jeden algoritmus pochází z druhého nebo naopak. Detekce duplikátů je nepřesná disciplína. Není přesně definována hranice, od které můžeme považovat dva algoritmy za shodné či rozdílné – hranici je nutno odhadnout. Pokud řekneme o dvou algoritmech, že si jsou podobné, neznamená to však, že se jedná o plagiátorství. Za plagiát označíme dílo, které vydává za své někdo jiné, než autor samotný. Tyto pojmy spolu úzce souvisí. Pokud různí programátoři řeší stejnou problematiku, je více než pravděpodobné, že některé algoritmy budou podobné. Je tedy obtížné, ne-li nemožné rozlišit plagiát od podobného řešení. Za plagiát bude v systému označen jakýkoli algoritmus, u kterého bude podobnost vyšší než přesně definovaná hranice. Některé porovnávané dvojice zcela určitě budou klasifikovány jako falešně pozitivní nebo falešně negativní. Vše bude záležet na separačním kritériu, hranici, dle které je dvojice klasifikována za plagiát či nikoli.

Obrázek 8: Klasifikační hranice

Obrázek 8 zobrazuje důsledek hranice klasifikace. Pro originály i duplikáty uvažujme standartní Gaussovo pravděpodobnostní rozložení. Pokud hranici zvýšíme (zpřísníme porovnávání) bude odhaleno více plagiátů, ale také bude více porovnávaných dvojic označeno falešně pozitivně. Při snížení hranice je falešná klasifikace potlačena, ale spolu s ní i úspěšnost odhalení duplicitní algoritmů.

Podobnost lze vhodně popsat vzdáleností. Pokud je vzdálenost mezi zkoumanými objekty menší než u jiných dvojic, jsou si objekty podobnější. Jednoduše lze určit vzdálenost mezi dvěma řetězci. Určuje kolika kroky, lze z jednoho řetězce zkonstruovat druhý, za použití operací smazání, vložení nebo přepsání znaků. Pokud by se na algoritmy pohlíželo jako na textové řetězce, určení vzdálenosti by nebylo složité vypočítat. Výsledek lze velice jednoduše ovlivnit. U některých programovacích jazyků nezáleží na pořadí určitých bloků kódů nebo na názvech proměnných. Využitím moderních IDE je možné docílit velké vzdálenosti mezi řetězci použitím například `refaktorování`[9]. Jak tedy určit vzdálenost mezi zdrojovými kódy? Problematika vyžaduje porozumění danému jazyku – neexistuje univerzální algoritmus pro všechny programovací jazyky. Každý jazyk má odlišnou syntaxi.

4.5.1 Plugin pro detekci plagiátorství programovacího jazyka

V systému CoDiAna je dostupný modul detekující duplicitní řešení v programovacím jazyce Java. Důležité je identifikovat základní prvky jazyka (třídy, metody, cykly, podmínky, inicializace, přiřazení, …). Plugin využívá knihovnu třetí strany pro syntaktickou analýzu [8] (parsování kódů). Zdrojový kód je za pomocí knihovny analyzován a jsou vytvořeny objekty základních prvků ve stromové struktuře. Plugin poté, za pomoci návrhového vzoru `Visitor`, navštíví vybrané základní prvky a vytvoří nový objekt, který obsahuje důležité informace k porovnávání. Plugin zpracuje zdrojové kódy obou řešitelů a porovná výsledné objekty. Při porovnávání jsou hledány nejpodobnější třídy a rozhraní obou řešení studentů. Určení podobnosti tříd a rozhraní závisí na modifikátorech objektů, komentářích a především na podobnosti metod. Ta je určena na základě vzdálenosti sekvence příkazů jednotlivých metod. Vy výsledku jsou mezi sebou porovnávány všechny třídy a rozhraní, všechny metody těchto objektů a všechny příkazy těchto metod.

Vzdálenost sekvence příkazů je určena obdobně jako u řetězců. Základem je `Levenshteinův algoritmus` [5] pro určení vzdálenosti řetězců, který je ovšem upraven. Modifikovaný algoritmus neporovnává řetězce, ale sekvence příkazů. Na

[9] Proces provádění změn ve zdrojovém kódu, aniž by byla ovlivněna funkcionalita programu.

každý příkaz je pohlíženo jako na znak řetězce – celá sekvence tedy tvoří slovo. Za každou operaci vložení, smazání nebo záměnu je udělena penalizace, která ve výsledku tvoří vzdálenost. Při porovnávání je brána v potaz podobnost příkazů. U významově podobných příkazů je udělena nižší penalizace. Například penalizace příkazů `For` a `Foreach` bude nižší než u příkazů `For` a `If`. Složitost porovnání dvou zdrojových kódů je určena dle vztahu:

$$P(x,y) = \sum_{i=0}^{n_1} \sum_{j=0}^{n_2} \sum_{k=0}^{m_{1,i}} \sum_{l=0}^{m_{2,j}} p_{1,i,k} * p_{2,j,l}, \; kde$$

$$n_i \cdots počet\ tříd\ řešitele\ i \qquad (2)$$

$$m_{i,j} \cdots počet\ metod\ třídy\ j, řešitele\ i$$

$$p_{i,j,k} \cdots počet\ příkazů\ metody\ k, třídy\ j, řešitele\ i$$

4.5.2 Neporovnatelná řešení

Sbírka všech řešení může obsahovat řešení v různých programovacích jazycích. Proces detekce duplicitních řešení mezi sebou porovnává každou dvojici řešení. Exekutivní aplikace přeskočí kontrolu dvojice, vyskytne-li se případ, ve kterém mají být porovnána řešení, která nemají stejný programovací jazyk. Důvod je zřejmý, pokud se řešení různí v programovacím jazyce, je pravděpodobnější, že první řešení nevychází z druhého řešení. Stejně tak, jako jsou přeskočeny dvojice s nestejným programovacím jazykem, jsou ignorovány i dvojice, ve kterých jedno z řešení nelze syntakticky analyzovat nebo kde jsou nedostupné zdrojové kódy. Tyto jevy mohou nastat, například manuální změnou souborů na serveru.

Syntaktická analýza zdrojových kódů jazyku Java využívá kompilační knihovnu [8]. Tato knihovna je však prozatím pro verzi 1.5. Může se tedy stát, že syntaxe novějších prvků vyšších verzí jazyka Java nebude akceptována a celé řešení bude neanalyzovatelné.

4.5.3 Optimalizace problému

Výpočetní složitost detekování duplicitních řešení je exponenciálního řádu v závislosti na počtu řešení. Počet operací nelze snížit, je nutné provést $\frac{1}{2}(n^2 - n)$ kroků pro n řešení. Abychom mohli optimalizovat počet nutných operací k porovnávání všech řešení, je nutné hlouběji rozebrat tuto problematiku. Celý proces porovnání jedné dvojice se skládá z následujících kroků. Jedná se o:

- sběr zdrojových kódů řešení,
- syntaktická analýza zdrojových kódů řešení,
- určení podobnosti mezi řešeními.

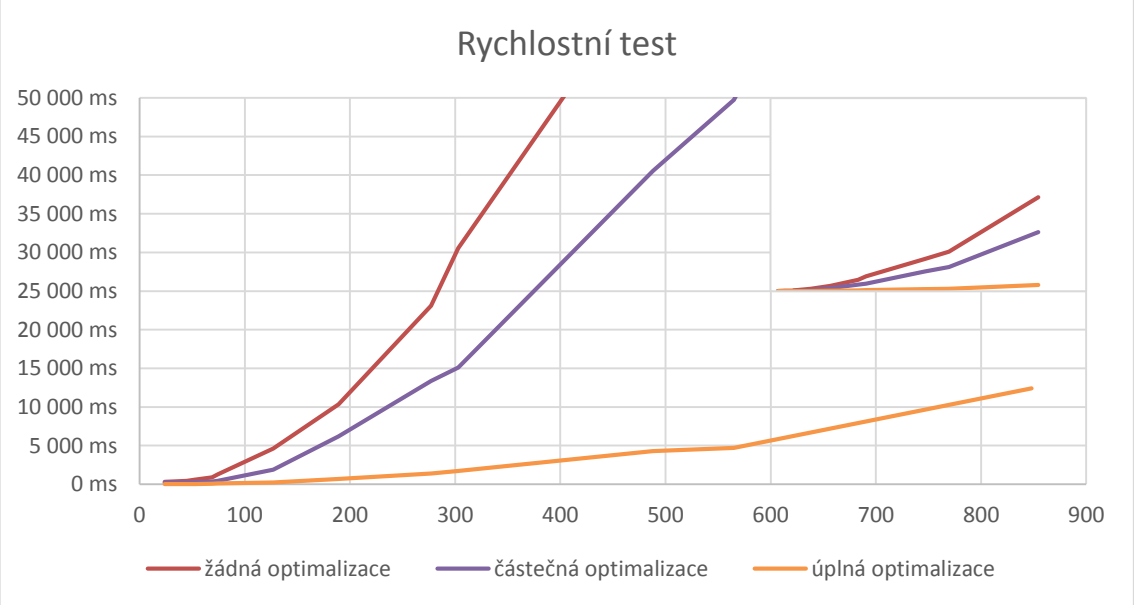
Pro každé porovnání je nutno provést tyto kroky. Akce sběru zdrojových kódů lokalizuje všechny potřebné soubory, které jsou součástí řešení. Ty jsou pak předány k syntaktické analýze. Po analýze algoritmů je vytvořen objekt, který reprezentuje všechny zdrojové kódy a jejich strukturu. Když jsou zkonstruovány oba objekty pro obě řešení, jsou mezi sebou porovnány. Byly prozkoumány tři přístupy, kterými jsou řešení předzpracována a následně porovnána. Shrnutí optimalizačních metod zobrazuje Tabulka 2. Optimalizační metody jsou:

1) Žádná optimalizace
   - Každá dvojice se před porovnáním vždy zpracuje. Dochází tedy ke zpracování stejného řešení několikrát. Tento způsob využívá minimální množství paměti, ale doba trvání zpracování, zvláště při větším počtu řešení, je velice vysoká.
2) Částečná optimalizace
   - Tento způsob předzpracuje jedno řešení a porovná jej se všemi ostatními řešeními. Paměťová náročnost je stále minimální a doba trvání má lepší výsledky.
3) Úplná optimalizace
   - Na začátku jsou předzpracována všechna řešení úlohy. V dalším kroku jsou řešení porovnána s využitím všech předzpracovaných objektů. Poslední způsob má největší paměťové nároky, ale časová náročnost je mnohokrát nižší v porovnání s předchozími způsoby. Při větším počtu porovnávaných řešení je potřebný čas i desetkrát nižší než v ostatních případech.

Tabulka 2: Porovnání optimalizačních metod

	Žádná optimalizace	Částečná optimalizace	Úplná optimalizace
paměťová náročnost	nízká	střední	vysoká
časová náročnost	vysoká	střední	nízká

Ve výsledku bylo zvoleno řešení `úplné optimalizace`. Při rozhodování bylo přihlíženo nejenom k časovým a paměťovým výsledkům. Důležité také je, kolik řešení se porovnává a jak složitá řešení jsou (jakou mají velikost). Pokud je počet řešení v dané úloze relativně malý, není paměťová rozdílnost tak významná. Z výsledků, které byly naměřeny za rok 2012, vyplývá, že průměrný počet studentů v dané úloze je 33. Maximální počet studentů v úloze nepřekročil hranici 60. Obrázek 9: Porovnání optimalizačních metod zobrazuje výsledky rychlostního testu.



Obrázek 9: Porovnání optimalizačních metod

Zvolené východisko v systému CoDiAna využívá dočasné paměti na předpočítání a udržuje v paměti všechny objekty, který jsou následně porovnány se všemi ostatními. Máme-li tedy úlohu, které obsahuje n řešení, postup je následující. V prvním kroku jsou předpracována všechna řešení ($ř_1$ až $ř_n$). Následně jsou určeny podobnosti mezi všemi řešeními. To je realizováno následovně: Nejprve je porovnáno řešení $ř_1$ se řešeními $ř_2$ až $ř_n$, dále řešení $ř_2$ s řešeními $ř_3$ až $ř_n$. Algoritmus končí porovnáním dvojice $ř_{n-1}$ a $ř_n$.

5 Bezpečnost systému

Velice důležitá je otázka bezpečnosti. Každý systém by měl být bezpečný, je nutno eliminovat co nejvíce míst v systému, které může ať už běžný uživatel nebo někdo jiný využít. Čím více prostoru dostane uživatel, tím je systém zranitelnější. Systém CoDiAna se zabývá spouštěním uživatelských zdrojových kódů, tzn., že libovolný uživatel může zaslat potenciálně nebezpečný zdrojový kód. Je mnoho oblastí, které může útočník zneužít. Jedná se například o neoprávněnou manipulaci s daty (odposlouchávání, krádež, smazání, …) nebo odstavení výpočetní jednotky. Útočník také může využít systém pro útoky na jiné stanice prostřednictvím sítě (odcizení dat, DDOS útoky, nebo další). Systém CoDiAna tedy musí obsahovat prostředky, které všechny tyto útoky znemožní.

Systém CoDiAna se skládá ze dvou hlavních částí. Modul CoDiAna zpracovává požadavky od uživatelů. Tyto požadavky mohou být potencionálně nebezpečné. Při vývoji systému byly využívány bezpečnostní prvky systému Moodle. Jedná se zejména o komunikaci s SQL databází (kontrola parametrů, apod.). Další bezpečností prvek je ošetření přijatých dat. V modulu je také využíván systém pravomocí a oprávnění, který omezuje vykonání určitých operací pro definované role. Exekutivní aplikace navíc vyžaduje bezpečnostní prvky související s omezením spouštěných procesů, které jsou dále popsány.

5.1 Izolace stanice

Bezpečnost v systému CoDiAna je založena na omezení práv jistého uživatele. V systému je vytvořen uživatel, který má minimální pravomoci a má velice omezené prostředky. Základní bezpečnostní opatření, které znemožní napadení jiných stanic, je zakázání internetového spojení pro tohoto uživatele. Všechny bezpečnostní hrozby se nadále týkají pouze současné stanice, pokud je řádně izolovaná od ostatních stanic. Další užitečné omezení je znemožnit uživateli přístup do jiného adresáře než do svého domovského adresáře [3].

5.2 Odmítnutí služeb

Všechna další bezpečnostní opatření už budou více specifická a nebudou pokrývat tak širokou oblast jako předchozí. Pro každý spuštěný proces v rámci zpracování řešení je definován globální časový limit. Tento čas označuje dobu, po kterou bude umožněno procesu vykovávat svou činnost. Po této době obdrží proces signál `kill`, který proces definitivně ukončí. Stejně, jako je proces omezen časově, je omezen i paměťově, opět globální hodnotou tentokrát však paměťového limitu. Dalším důležitým opatřením je omezení procesů daného uživatele. Jednou z podob jsou útoky tzv. `Fork bomby`, kde proces generuje nové podprocesy a ty opět nové podprocesy. Omezení počtu podprocesů zamezuje provádění `fork` bomb, které by ve výsledku vyčerpaly volnou paměť nebo procesorový čas. Tato opatření zabezpečují oblast `DoS`[10] [11].

Pokud jsou omezeny časové a operační paměťové prostředky, je také nutné omezit úložiště. Uživatel má opět omezené prostředky pro práci se soubory. Spuštěný proces může pouze generovat výstup nebo chybový výstup a načítat vstup. Práci s ostatními soubory je vhodné v produkční verzi z bezpečnostních důvodů omezit. Oba výstupní soubory mají opět limitovanou maximální velikost, neboť by mohl uživatel výpisem na standardní výstup vytvářet neomezeně velké soubory.

5.3 Další bezpečnostní opatření

Pro systém CoDiAna byl vytvořen testovací server obsahující kopii E-learningového portálu TUL. Na tomto serveru byla testována všechna bezpečnostní opatření. V produkční verzi je vhodné přidat další pokročilejší bezpečnostní prvky, které zabezpečí cílový server.

5.3.1 Změna kořenového adresáře

Výše uvedená opatření zajistí izolaci stanice od všech ostatních a zabrání případnému vyřazení stanice. Potencionální útočník však stále může zjistit strukturu serveru obsahující Exekutivní aplikaci, nebo alespoň části serveru. V produkční verzi je vhodné změnit kořenový adresář příkazem `chroot`.

[10] Odmítnutí služby

Spuštěnému procesu je tak znemožněno nahlížet do jiných adresářů. Realizace tohoto opatření může být provedena pomocí balíčku programů `Jailkit`. Tyto programy zjednodušují konfiguraci pro izolaci určitých aplikací nebo uživatelů. Spuštěný proces má při použití `Jailkit` přístup pouze do určitých adresářů a pouze ke zvoleným binárním aplikacím [10][13].

5.3.2 Virtualizace

Pokud by útočník přes všechna bezpečnostní opatření našel zranitelné místo systému, je důležité, aby byly minimalizovány škody. Jedním z řešení je virtualizace, kdy je vytvořen virtuální stroj. Na tomto stroji je pak spuštěn potencionálně nebezpečný proces. Při útoku na stanici je v horším případě vyřazen pouze virtuální stroj, který může být znovu obnoven. Ovšem kombinací chybného nastavení virtuálního stroje (např. firewall) a zručností útočníka, lze vyřadit i fyzický server. Nikdy není zaručeno, že je aplikace zabezpečená proti všem útokům [14].

6 Testování systému

Celý systém byl testován již v průběhu jeho vytváření. Modularita systému umožňovala postupné testování nově vytvořených části systému. Po zkompletování systému byly provedeny rozsáhlé testy, které lze rozdělit na dvě skupiny: testování funkčnosti a testování bezpečnosti.

6.1 Testování funkčnosti systému

Tato skupina testů zkoumá korektní reakci systému na určitou operaci. Operace v systému může být například `vytvoření úlohy` nebo `odevzdání řešení úlohy`. Při vyvolání této operace bylo testováno, zda jsou související data operace načtena v příslušném formátu a zda jsou ošetřeny všechny krajní možnosti algoritmu. V průběhu testování byly vytvořeny desítky algoritmických úloh a na tyto úlohy byly zaslány stovky řešení. Byly odhaleny chyby související s prostředky na serveru (oprávnění, omezení procesů, apod.) a programováním (například v krajních případech algoritmů). Testování bylo prováděno na serveru obsahující kopii E-learningového portálu TUL.

6.2 Testování bezpečnosti systému

Rozsáhlejší systém může obsahovat více chyb než méně komplexní systém. Testování bezpečnosti probíhalo systematicky. Nejprve byl testován modul CoDiAna. Byl testován přístup k nepovoleným stránkám nebo provádění nepovolených operací. Na závěr byly testovány vstupy formulářů.

Testování Exekutivní aplikace bylo složitější, neboť má potencionální útočník více prostoru pro napadení serveru. Při testování byly vytvořeny škodlivé skripty, které byly zaslány na zpracování. Byla testována opatření proti vyčerpání prostředků na serveru, přístupu k internetu a práce se soubory. Z testovaných případů pouze jeden skript prováděl neoprávněné operace – práci se soubory v domovském adresáři uživatele s omezenými oprávněními. Tento problém může být vyřešen změnou struktury soubory na serveru, virtualizací nebo také změnou kořenového adresáře. Ostatní testované případy proběhly v pořádku (proces byl buď ukončen, nebo nebyla operace provedena).

Závěr

V rámci této práce byl vytvořen systém CoDiAna. Systém umožňuje zautomatizování procesu testování dovedností studentů v předmětech s tématikou programování a algoritmizace. Pomocí tohoto systému lze vytvářet a spravovat algoritmické úlohy vytvořené pedagogem – je nutné ve specifikaci úlohy přiložit vstupní a výstupní soubory. Výsledný systém zjednodušuje vytváření zmíněných souborů – obsahuje generátor vstupních souborů, který slouží pro tvorbu a editaci jednoduchých vstupních souborů úlohy. Výstupní soubor je možné vygenerovat zpracováním referenčního řešení od pedagoga. Touto operací je také možné stanovit časovou a paměťovou náročnost úlohy.

Po zadání algoritmické úlohy mohou studenti odevzdávat svá řešení, která jsou systémem vyhodnocena – je ověřena jejich korektnost a následně navržena výsledná známka odevzdaného řešení. Řešením může být soubor nebo dokonce skupina souborů obsahující zdrojové kódy, řešící zadanou úlohu. Velkou výhodou systému je nezávislost na konkrétním programovacím jazyku. Systém umožňuje odevzdávat řešení v libovolném jazyce, který je serverem podporován. Systém CoDiAna dále nabízí prostředky pro určení podobnosti odevzdaných řešení a následnou detekci plagiátů. Nastavení algoritmických úloh je detailní, lze nastavit mnoho argumentů úlohy, například hodnotící časové a paměťové hranice, pomocí kterých je vypočtena výsledná známka. Dále je možné nastavit metody pro porovnání vygenerovaných výstupních souborů nebo také způsoby výběru hodnoceného řešení z více korektních řešení studenta. Systém nabízí celkem tři metody porovnání výstupních souborů, které se různí v přísnosti preciznosti porovnávaných souborů. Pedagog může dále specifikovat ukázkový vstup a na něj reagující výstup úlohy. Proces zpracování řešení zahrnuje měření hodnotících veličin a následné porovnání vygenerovaného výstupního souboru s referenčním výstupním souborem.

Výsledný systém se skládá ze dvou hlavních částí. Obě části systému mohou být na oddělených počítačových serverech, komunikace je poté zajištěna SSH spojením. První část je modul CoDiAna systému Moodle, která slouží pro správu algoritmických úloh, odevzdávání řešení a prohlížení výsledků úloh. Tyto operace

je možné realizovat intuitivním uživatelským grafickým rozhraním. Vytvořený modul nabízí prostředky pro správu souborů (přiložení referenčních souborů, odevzdání řešení) a prohlížení výsledků úlohy v textové nebo také grafické podobě. Pro tento modul byla vytvořena „Příručka pro uživatele modulu CoDiAna“, ve které jsou popsány možnosti modulu pro pedagogy i studenty.

Další částí systému je tzv. Exekutivní aplikace, která slouží pro zpracování požadavků od modulu CoDiAna. Zajišťuje kompilaci, spuštění řešení a detekci duplicitních řešení. Vhodnou strukturou aplikace je umožněno dynamické načítání rozšiřujících modulů, které přidávají podporu určitého programovacího jazyka. Jedná se o podporu zpracování programovacího jazyka nebo syntaktickou analýzu programovacího jazyka. V průběhu řešení byly vytvořeny moduly umožňující plnou podporu jazyka Java. Dále byla přidána podpora zpracování programovacích jazyků C, C++ a Python. Pro Exekutivní aplikaci byla vytvořena „Příručka pro vývojáře systému CoDiAna“ obsahující pokyny pro tvorbu nových modulů.

Při návrhu bylo dbáno na bezpečnost celého systému. Zpracování řešení je vesměs spouštění neznámého kódu a může být velice nebezpečné. V systému jsou zabudované bezpečnostní prvky, které izolují spuštěný proces od ostatních pracovních stanic a dokonce od samotné adresářové struktury serveru. Omezeny jsou také časové a paměťové nároky.

Výsledný systém CoDiAna je provozuschopný a nahrazuje systém vytvořený v rámci bakalářské práce řešící obdobnou problematiku. Nově vytvořený systém je integrován do školního e-learningového systému a nabízí modernější a přívětivější uživatelské rozhraní, vytvořené za pomocí novodobých technologií. Struktura a modularita systému dává prostor dalším rozšířením. Mohou být definovány další způsoby, které podpoří praktickou část předmětů s programovací tématikou. Jedním ze způsobů může být definice rozhraní, které musí studenti implementovat. Lze také vyžadovat nebo zakázat určité prostředky ve specifikaci úlohy.

Seznam použité literatury

[1] BARRETT, Daniel J., Richard E. SILVERMAN a Martin BLAŽÍK. *SSH: Kompletní průvodce*. Brno: Computer Press, 2003. ISBN 80-7226-852-X.

[2] *Dokumentace systému Moodle pro vývojáře:* Developer documentation. Moodle [online]. 2013, 2013-12-10 [cit. 2014-03-09]. Dostupné z: docs.moodle.org/dev/Developer_documentation

[3] Disable internet access for particular user in Ubuntu. *Ubuntu Geek: Ubuntu Linux Tutorials* [online]. 2008, 2008-01-09 [cit. 2014-03-10]. Dostupné z: www.ubuntugeek.com/disable-internet-access-for-particular-user-in-ubuntu.html

[4] G, Mike. *BASH Programming: Introduction HOW–TO* [online]. 2000, Thu Jul 27 09:36:18 ART 2000 [cit. 2014-03-09]. Dostupné z: www.tldp.org/HOWTO/pdf/Bash-Prog-Intro-HOWTO.pdf

[5] HALDAR, Rishin a MUKHOPADHYAY. Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. Calcutta, India, 2011. Dostupné z: arxiv.org/ftp/arxiv/papers/1101/1101.1232.pdf

[6] HYBŠ, Jan. *Serverová část systému pro podporu praktické výuky programování*. Liberec, 2012. Bakalářská práce. Technická univerzita v Liberci. Vedoucí práce doc. Ing. Jiřina Královcová, Ph.D.

[7] Chart.js Documentation: Everything you need to know to create great looking charts using Chart.js. *Chart.js: HTML5 Charts for your website* [online]. 2014 [cit. 2014-04-22]. Dostupné z: www.chartjs.org/docs/

[8] Javaparser: Java 1.5 Parser and AST. *Google Code* [online]. [2009] [cit. 2014-03-09]. Dostupné z: https://code.google.com/p/javaparser/

[9] JSch: Java Secure Channel. *JCraft: Code the Craft, Craft the Code* [online]. [2012] [cit. 2014-03-12]. Dostupné z: www.jcraft.com/jsch/

[10] PAVLÍČEK, Martin. Chroot prostředí. *AbcLinuxu.cz: Linux na stříbrném podnose* [online]. 2013, 2013-12-16 [cit. 2014-04-11]. Dostupné z: www.abclinuxu.cz/clanky/bezpecnost/chroot-prostredi-i

[11] Setting Shell Limits for the Oracle User. *Red Hat: The World's Open Source Leader* [online]. [2009] [cit. 2014-03-09]. Dostupné z: https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Tuning_and_Optimizing_Red_Hat_Enterprise_Linux_for_Oracle_9i_and_10g_Databases/chap-Oracle_9i_and_10g_Tuning_Guide-Setting_Shell_Limits_for_the_Oracle_User.html

[12] Speed of INSERT Statements. *MySQL: The world's most popular open source database* [online]. [2007] [cit. 2014-01-19]. Dostupné z: dev.mysql.com/doc/refman/5.0/en/insert-speed.html

[13] ŠTRAUCH, Adam. Jailkit: snadné chrooování služeb. *Root.cz: Informace nejen ze světa Linuxu* [online]. 2008, 2008-11-28 [cit. 2014-04-11]. Dostupné z: www.root.cz/clanky/jailkit-snadne-chrootovani-sluzeb/

[14] Virtualizace: Fenomén dneška. *Trask Solutions: Integrace, zavádění a správa IT řešení* [online]. [2009] [cit. 2014-03-09]. Dostupné z: www.trask.cz/virtualizace-fenomen-dneska

[15] *Zip4j: Java library to handle Zip files* [online]. 2010 [cit. 2014-03-09]. Dostupné z: www.lingala.net/zip4j/index.php

A Obsah přiloženého CD

Přiložené CD obsahuje všechny nutné soubory pro instalaci systému CoDiAna. Obsah přiloženého CD je organizován do adresářové struktury, významy jednotlivých adresářů jsou následující:

- Adresář `dokumentace`
  - obsahuje elektronickou podobu této práce. Práce je uložena ve formátech `pdf`, `docx` a `doc`.
  - obsahuje originální zadání ve formátu `pdf`.
- Adresář `prirucky`
  - obsahuje elektronickou podobu příručky pro vývojáře modulu CoDiAna a příručky pro uživatele systému CoDiAna. Obě příručky jsou k dispozici ve formátech `pdf`, `pptx` a `ppt`.
- Adresář `modul_CoDiAna`
  - obsahuje všechny soubory, který jsou nutné pro instalaci modulu CoDiAna do systému Moodle.
- Adresář `exekutivni_aplikace`
  - Podadresář `bin` obsahuje spustitelnou Exekutivní aplikaci Java a všechny požadované knihovny a základní konfigurační soubor.
  - Podadresář `src` obsahuje zdrojové kódy Exekutivní aplikace Java a vytvořených modulů.
  - Podadresář `kostry_modulu` obsahuje archivy, ve kterých jsou uloženy ukázkové zdrojové kódy pro tvorbu nových modulů.